NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TWO NEW NEAREST NEIGHBOR CLASSIFICATION RULES

by

Ciril Karo

September 1998

Thesis Advisor:	Samuel E. Buttrey
Second Reader:	Harold J. Larson

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1998	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE TWO NEW NEAREST NEIGHBOR CLASSIFICATION RULES	5. FUNDING NUMBERS
6. AUTHOR(S) Ciril Karo	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Nearest-Neighbor (NN) classification is a non-parametric discrimination and classification technique. In NN classification a test item is compared by some similarity measure of its multiple variables (usually a distance metric) with all the items in a training set. The class of the item to which it is most similar can be used as an indication of the class of the test item. In other words, the test item is assigned the class of its nearest neighbor. A key extension is the case when $k > 1$ nearest neighbors (k-NN) are examined with the classification usually being made based on a plurality.

NN classification is used in many fields, including for example the field of Pattern Recognition. Applications include tasks like speech recognition by a computer, medical data interpretation and diagnosis, or the interpretation of remote sensing imagery from satellites. Military applications of the technique include any situation where automated recognition is required.

This thesis proposes two new NN rules that are intended to improve classification accuracy. The rules are tested against baseline classification methods in common use with a variety of data sets. One method shows improvement over the baseline methods in most of the data cases examined.

14. SUBJECT TERMS Nearest Neighbor Classification, Discrimination, Pattern Recognition	15. NUMBER OF PAGES 86
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

TWO NEW NEAREST NEIGHBOR CLASSIFICATION RULES

Ciril Karo
Major, Australian Army
BSc(Hons), University College (Australian Defence Force Academy),
University of New South Wales, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1998

Author: Ciril Karo
Ciril Karo

Approved by: [illegible]
Samuel E. Buttrey, Thesis Advisor

[illegible]
Harold J. Larson, Second Reader

Richard E. Rosenthal
Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

Nearest-Neighbor (NN) classification is a non-parametric discrimination and classification technique. In NN classification a test item is compared by some similarity measure of its multiple variables (usually a distance metric) with all the items in a training set. The class of the item to which it is most similar can be used as an indication of the class of the test item. In other words, the test item is assigned the class of its nearest neighbor. A key extension is the case when $k > 1$ nearest neighbors (k-NN) are examined with the classification usually being made based on a plurality.

NN classification is used in many fields, including for example the field of Pattern Recognition. Applications include tasks like speech recognition by a computer, medical data interpretation and diagnosis, or the interpretation of remote sensing imagery from satellites. Military applications of the technique include any situation where automated recognition is required.

This thesis proposes two new NN rules that are intended to improve classification accuracy. The rules are tested against baseline classification methods in common use with a variety of data sets. One method shows improvement over the baseline methods in most of the data cases examined.

THESIS DISCLAIMER

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Nearest-Neighbor (NN) classification is a non-parametric discrimination and classification technique. In NN classification a test item is compared by some similarity measure of its multiple variables (usually a distance metric) with all the items in a training set. The class of the item to which it is most similar can be used as an indication of the class of the test item. In other words, the test item is assigned the class of its nearest neighbor. A key extension is the case when $k > 1$ nearest neighbors (k-NN) are examined with the classification usually being made based on a plurality.

NN classification is used in many fields, including for example the field of Pattern Recognition. Applications include tasks like speech recognition by a computer, medical data interpretation and diagnosis, or the interpretation of remote sensing imagery from satellites. Military applications of the technique include any situation where automated recognition is required.

Nearest-neighbor classification is a computationally expensive technique. Following initial developments in the field there was considerable research effort directed towards reducing the computational load of the method. Typical approaches involved preprocessing the training set to reduce the number of distance calculations required without reducing the accuracy of the classifier. Whilst faster computers and more memory reduce this concern, the method remains difficult to do in real time.

The idea of being able to augment the NN decision rules with additional embellishments to improve accuracy is another area of research activity. An example of such is the intuitively appealing distance weighting idea. That is, the closer nearest neighbor rated a higher proportion of the classification vote. An interesting observation was that the improvements in accuracy for such (rule) embellishments are generally minor.

This thesis proposes two new NN rules that are intended to improve classification accuracy. The first rule, called knn-in-leaf, involves finding regions of class purity in the feature space and conducting *k*-NN inside those spaces. The intuition behind the approach is that if the region is "class pure," then we will have more confidence in classifications made in that region. It is hoped that this improved confidence will manifest itself as a lower misclassification rate. The question of how to split the region is difficult. We chose a classification tree as a simple splitting tool with which to prove the concept. In effect the new rule is *k*-NN nested inside a classification tree.

The second rule is called r.d, which is short for radial distance. Rather than selecting an optimal number *k* of nearest neighbors to consider, we instead consider all the nearest neighbors inside a local radius. The optimal radius is chosen by a search over a range of appropriate radii using cross-validation in the training set. Once the radius is set, each test item is compared to all its nearest neighbors in the local neighborhood thus created. This technique allows the neighbors in a dense part of the feature space to provide more information to the classification decision. Where the region is sparse the number of nearest neighbors will be fewer, but it is hoped it will be more accurate than *k*-NN, as nearest neighbors more distant than the optimal distance would have been used in a *k*-NN classification.

The proposed new rules are tested against the baseline classification methods of *k*-NN and classification trees with a variety of data sets. The measure of effectiveness for the rules was their misclassification rate. In all data sets, knn-in-leaf was able to show some improvement over the classification tree. Of the four methods, knn-in-leaf was the best in six out of 10 instances. r.d was the best in one case, with *k*-NN being the best in the remainder. The improvements in misclassification rate for knn-in-leaf were in the range of 1-3%. The r.d improvement was 0.5%.

It is recommended that further research be done on the knn-in-leaf method. In particular the method of splitting the feature space into regions of class purity/impurity should be investigated more thoroughly. There are also many improvements to be made to the basic rule as it exists now. It is not recommended that any further research effort be expended on r.d.

I. INTRODUCTION TO NEAREST NEIGHBOR CLASSIFICATION

A. BACKGROUND

Nearest-Neighbor (NN) classification is a non-parametric discrimination and classification technique that has found wide use and acceptance in the last 20 years. The approach in NN classification is to hypothesize that a test item can be compared by some similarity measure of its (multiple) variables (usually some form of distance metric) with all the items in a training set. The class of the item to which it is most similar is used to define the class of each test item. In other words, the test item is assigned the class of its nearest neighbor. Informal definitions include the phrases "birds of a feather flock together" or "judge a person by the company he keeps" (Dasarathy, 1990). A key extension to this basic idea employs $k > 1$ nearest neighbors (k-NN), with the classification usually being based on a plurality.

This chapter will introduce the concept of nearest-neighbor classification. A short survey of key developments in the field will be given followed by some areas of past and current research. The uses of the nearest-neighbor method will be examined, and in particular some military applications will be identified. Finally, the chapter will set out the structure of the remainder of this thesis.

B. INITIAL CONCEPTS

The initial concept developed out of pioneering work done by Fix and Hodges (1951) on non-parametric discriminant analysis. They did not define NN classification *per se*, but rather through their investigation of the consistency properties of non-parametric discrimination problems opened the door for further development of non-parametric discrimination rules. They defined the non-parametric discrimination problem at length and demonstrated the consistency properties of certain non-parametric discrimination procedures.

The formal development of NN rules as they are known now is ascribed to Cover and Hart (1967). Of importance in their paper was that they were able to derive the error rates for the NN method in terms of the Bayes probability of error. The Bayes probability of error is the rate at which errors would be committed (in the classification decision) if the true density and class priors (actual proportion of classes in the population) were known. In particular the upper bound on the error was shown to be at most twice the Bayes error. The interpretation ascribed to this result was that half of all the information available in an infinite training sample was contained in the nearest neighbor. The details of how they derived such error rates is not required here, but it suffices to say that the potential and popularity of the NN method received a boost (Dasarathy, 1990).

The final core development was the extension to k-NN by Patrick and Fisher (1970). This approach considered some number $k > 1$ of nearest neighbors to be examined with the classification based on a plurality. The method of k-NN will be examined in more detail later, as variations of this rule are the basis for one area of the current research.

C. FURTHER DEVELOPMENTS AND RESEARCH

Following the initial work on the NN rules, the developments in the field took three main tracks. The first involved efforts to reduce the computation required to find a nearest neighbor. The second was the issue of error estimation and the risk in using the technique, and finally and of most interest to this investigation were algorithmic developments. These will be outlined below in turn. An excellent summary of all these can be found in Dasarathy (1990) from which most of the following discussion was gathered.

1. Computational Concerns

A basic pseudo-code algorithm to find a single nearest neighbor using Euclidean distance is as follows:

```
Inputs:
Training set of N items and V variables and a known classification $N_C$;
(with each population class represented in the training items)
Test set of M items with V variables and unknown classification;
Output:
NN predicted Classification of each item in test set;
For each item (Test) in test set{
    Initialize current nearest neighbor to first item in training set;
    CurrentMinDist ← infinity;
    For each item (Trng) in training set{
        Dist ← 0 ;
        // Calculate square of Euclidean distance to test item as follows:
        For each variable j in V {
            Dist ← Dist + $(Trng_j – Test_j)^2$;
        }
        if (Dist < currentMinDist) {
            update incumbent nearest neighbor to Trng;
            current min distance ← Dist;
        }
    } // end for each training item
    return classification of incumbent nearest neighbor
} next test item
```

The worst-case complexity to find the nearest neighbor for a single item in a training set of size N, with each item having V variables, is on the order of N times V, written O(N*V). In itself this is not a large problem except that in the potential uses of this method the test set is generally of size M (>>N). The implication here is that if N is already large, then a very large number of calculations is required. Thus, at least in the initial period when the method was being developed the complexity issues were considerable. It should be noted that increases in computational power have (as in other areas of Math / Numerical Programming) not reduced the problems of computational complexity. The understanding of such issues is still crucial, as the result of better computers is bigger problems. NN classification is still a "large" problem (in terms of computation) and difficult to do in real time.

In the literature the approaches to reducing computational complexity took two general forms. The first considered that one could preprocess the data to reduce the number of distance calculations required. This involved sorting and/or ordering algorithms for the training set items so that the number of distance calculations in the classification phase would be minimized. The sorting involved various specialized data structures and search methods. A Branch and Bound algorithm proposed by Fukunaga and Narendra (1975) was typical of the type of ordering approach tried. The training set was preprocessed using a clustering scheme into disjoint subsets arranged in a hierarchical tree. The Branch and Bound algorithm then searched through the tree using decision rules at each node until the node of interest was found. The nearest neighbor was then found in that subset node. The obvious tradeoff here in terms of complexity was the amount of work required to preprocess the training set relative to the number of test items to be classified.

Another typical sorting approach involved careful selection of reference points or preprocessed representational sets which in effect provided a place to "break into" the training set and reduce the scope of the computations to just that local area.

The second major approach to the issue of computational complexity was training set reduction methods. If one could produce a minimum-sized consistent set of training items then one could classify just as accurately from the smaller set as from the larger. Obviously the smaller set required fewer calculations. A host of methods were proposed with names like Condensed Nearest Neighbor Rule (CNN), Reduced Nearest Neighbor Rule (RNN), Feature Space Partitioning (FSP), Prototype Nearest Neighbors (PNN) etc. The key facet of all these methods was to reduce the size of the training set without losing the information available in that training set. Reduction of the training set by one item should not greatly increase the probability of an incorrect classification for the set as a whole. Again a typical example of this scheme would examine the effect on misclassification rates if a training set item was removed or added and make a decision on its inclusion in the set. In the worst case a training set may not be reducible.

The push to reduce computational complexity was aided by developments in computer speed and technology. Faster computers pushed some problems to the background, but hardware developments such as the parallel computing systolic array (Dasarathy 1990) have made NN techniques more practical than ever. A second development is the cheap availability of large amounts of memory. Algorithms that carried out explicit search strategies often suffered from the need to repeat many steps or calculations for every test item. An approach by Broder (1990) explicitly stored valuable information about the search status in an auxiliary data structure that was constrained only by the amount of memory. Indications are that with the large amount of memory available, algorithms in future will tend to use more memory to gain computational time advantages.

2. Error Estimation

The issue of error estimation and the generalization properties have generated a host of papers, most of which are well beyond the scope of this thesis. The key issues are the risk associated with using a particular classification method and the convergence properties of the method.

It has been shown that for the basic 1-NN classifier in the infinite training set case, the maximum error is at most twice the Bayes error rate (Cover and Hart, 1967). Of course infinite-sized training set results are of academic interest only so similar results had to be developed for finite-sized sets. Cover (1968) showed the risk of a NN classifier in the finite sample case (of size n) would converge to the risk of the infinite case at a rate on the order of $1/n^2$. The result by Cover was extended by Peterson (1970) who showed that for a given level of risk, the optimal size of a required training set could be established.

Notwithstanding the above results the issues of error estimation in this paper relate purely to the empirical results observed. The error rates are expressed as misclassification rates where test items of known classification are used and the predicted classification is compared to the actual situation.

3. Algorithm Advances

Nearest-Neighbor classification started with a basic 1-NN classification rule. After that, the *k*-NN rule became popular. Since then there have been a host of investigations of different NN schemes with the aim being to improve the predictive ability of the rule. Since that is also the intention of this thesis some understanding of what has gone before is required.

The first algorithmic advances were various weighting schemes for the *k*-NN. A distance-based weighting scheme was proposed by Dudani (1976) for *k*-NN. His premise was that the confidence about the class of the test item compared to a training set item ought to be inversely proportional to the distance from that item. A variety of studies (Bailey and Jain (1978), Morin and Raeside (1981), Macleod, et al (1987)) followed this, with several suggesting modifications to the basic weighting function. Surprisingly the improvements in misclassification error rates generally were disappointing and some authors were able to show that there was little or no improvement over the basic *k*-NN classifier. Spawned from these investigations was further examination of what to do with ties (equal number of classes within the *k*-NN) in the *k*-NN classifier. Random or weighted random tie breaking appeared to be the most effective.

Brown and Koplowitz (1979) proposed that if the *a priori* probability of class membership is known and the training set does not exhibit the same relative proportions of the classes, then the single nearest neighbor should be weighted to take account of this discrepancy. In effect, if a class has disproportionately more items in the training set than the *a priori* probability would indicate, then the distance from the test item to the elements in that class is increased.

A further adaptation of *k*-NN by Tomek (1976) was to apply an integer threshold value k_1 ($0<k_1<k$) rather than a traditional plurality vote acceptance criterion. That is, a test item is assigned to a class only if there are at least k_1 members of that class among the k nearest neighbors. This then allowed the chance of a no classification or reject option, to be given if the number of votes for a class did not exceed the threshold.

O'Callaghan (1975) examined the definition of a neighborhood of a sample point in feature space. He examined the premise in *k*-NN that exactly *k* neighbors comprise the neighborhood regardless of their spread. In particular he proposed rules that added direction and distance constraints to the definition of a *k*-NN neighborhood. His rules, in essence, restricted the distance a neighbor could be from a test item and did not allow a neighbor to vote if that neighbor was occluded by another in the feature space. The distance aspect of his study is of interest in this thesis and is discussed more in Chapter II.

The next advancement was by Dasarathy and Sheela (1979) who reported the use of composite classifiers. They were able to optimally partition the feature space to suit the use of NN classifiers in one partition and some other different classifier in another. In particular the non-NN classifier would be one not requiring huge computation. The result was reduced computation whilst retaining the conceptual advantage of the NN classifier. This idea of multiple classifiers is of interest in this thesis as one of the schemes proposed uses two classification methods in a nested form.

The final aspect in the development of algorithms, which too many of the previous studies had ignored, was the very real problem of selecting the number *k* of nearest neighbors to use. The error studies to some extent considered the asymptotic qualities of the *k*-NN over a range of *k*, but fell short of proposing methods for trading off the increased accuracy often seen with a larger *k* versus computational complexity. It is considered that choosing a larger neighborhood (defined by the *k*), lessens the impact of any outliers. The classification is considered more robust, but there are problems with allowing *k* to get too large. The obvious one is that an excess of samples from another class may be gathered from outside of the local neighborhood of interest. It is generally expected that *k* will be much less than the size of the smallest class in the feature space. One simple and obvious approach is to select the *k* to use empirically from the data available (in the training set), by doing some cross-validation experiments with different *k*'s and selecting the one with the best properties. This is the approach taken in this thesis.

4. Modified Metrics

The basic NN rules generally assume the use of standard L_2-norm or Euclidean distance. Some researchers have questioned the appropriateness of this distance measure. A more generic method to choose the neighbors is the one that was expressed at the start of this chapter as "some measure of their similarity." If distance in Euclidean sense is not appropriate then some other rule needs to be employed. Research to date has focussed on finding a local area using Euclidean distance, and then using some modified rule in the local area using statistics of the system to choose the neighbors (Short and Fukunaga, 1981). Other aspects of Modified Metrics include the problem of how to find a distance when the variables of interest are categorical.

D. USES OF NEAREST NEIGHBOR CLASSIFICATION

Like most statistical techniques, NN classification does not have one best application. As a general classification technique NN classification can be used on many of the tasks for which any other classifier could be used. The choice of classifier is a task that will fall to a statistician who will trade off a range of problem and classifier attributes to achieve the best model or rule for the data.

1. Pattern Recognition

The most common area of use of NN rules is in the field of Pattern Recognition (PR). This discipline is simply described as the use of computers to perform automated recognition tasks. Pattern Recognition has grown as a discipline since the advent of computers. A typical PR task is illustrated by Optical Character Readers, such as those that read addresses and Zip Codes at the post office. Automated image processing applications are also popular. This could be the simple detection of differences in two photographs or X-rays, or it could involve development of a 3-D image from a 2-D map. Pattern Recognition also has non-visual uses. The techniques can be applied to survey or questionnaire data to ascertain a person's likelihood of having a disease, or ability to pay back a loan. In all cases, the PR algorithm will rely on some training data to which the item, image or questionnaire can be compared and a decision or classification applied. NN classification is one of the basic tools of PR algorithms.

2. Military Applications

The literature on actual uses of NN classification in military situations is almost non-existent. It could be expected that the PR applications discussed above will be in use, and it is easily surmised that many of the visual automated recognition-type tasks could easily be part of a wide range of military systems. Two concrete examples of the use of nearest neighbor techniques are given below.

Multi-target Aircraft Tracking: Nearest neighbor based filters are used for track development (Dasarathy, 1990).

Automated Recognition Systems: One data set in this thesis sought to classify sonar responses as either belonging to a mine (metal cylinder) or a rock. The original use of this data set was in developing a neural network classifier. The NN algorithm was used as a testing and tuning tool.

E. PURPOSE OF THIS THESIS

The intention of this thesis is to propose two new NN algorithms and to test their predictive ability against current classifiers. This thesis does not have one specific application; the general improvement in classifier accuracy is a topic of interest in many of the applications discussed above.

F. STRUCTURE OF THIS THESIS

Chapter II discusses the intuition behind better classification rules; in particular it describes in detail the two new methods proposed by this thesis and the rationale behind them. The chapter then discusses the testing procedure and techniques. It includes a description of baseline classification methods and the data sets used to test the new methods against the baseline methods.

Chapter III contains results and discussion. The results are given by data set, with each of the baseline and new methods being ranked in their performance. The Measure of Performance is simply the misclassification rate, although other aspects of classifier performance are also discussed. Finally, Chapter IV will raise issues for further investigation and present summary conclusions. Two appendices contain raw results and the S-Plus code used to obtain the results.

II. NEW CLASSIFICATION RULES

A. PROPOSED NEW CLASSIFICATION RULES

In the literature on NN classifiers reviewed for this thesis, one characteristic has stood out. It was that generally the basic plurality decision *k*-NN rule was the most appropriate one to use. That is, the improvement from embellishing the rule with a weighting scheme or rejection rule was minor compared to misclassification error rates of the basic *k*-NN rule. Of course most of the authors could find a data set that would show some positive trends for their methods, but as was amply seen in the case of distance weighting, the results have been inconsistent (Dudani (1976), Bailey and Jain (1978), Morin and Raeside (1981), Macleod, et al (1987))

The problem with this observation is that our intuition tells us that weighting schemes should work. In the case of distance weighting, the whole premise of NN classification is that "one is like one's neighbor." Why shouldn't the closer neighbors be more important in that decision? Thus, the idea of being able to embellish *k*-NN with some smarter rule is one that continues to provide researchers in the field plenty of opportunities to propose algorithms.

Following on the intuition route, this thesis examines two algorithms that have intuitive appeal to see if they are able to improve upon the basic accuracy of other classification schemes. The remainder of this section explains the two methods along with the rationale for their selection. The remainder of the chapter outlines the methods of testing to be used for the algorithms.

1. Algorithm 1. Knn-in-leaf

This method involves building a classification tree (of size determined in the algorithm), then conducting *k*-NN inside the terminal leaves of the tree. (For a good introduction to classification trees, see Breiman, et al (1984)).

The method grew from the intuition that if we were able to find regions of class purity in the feature space, then surely we would be more confident of classifications (to the pure class) made in those regions. It was also hoped that the overall misclassification rate would be improved in those regions. The question then arose as to how to find such regions of class purity. The obvious approach to such a feature space splitting would have been to use a clustering algorithm, but since those methods generally use distance-based rules, it was considered that no new information would be available to the *k*-NN classification decision. The next thought was to try to screen the data on a few of the variables to produce local neighborhoods of higher class purity. How to choose the variables to split the feature space was the next logical question. The topic of variable selection is very large and of importance in all aspects of statistical model building, but as a simple proof of concept example, it was decided that the feature space splitting could be achieved by the use of an existing tool, the classification tree.

In a classification tree 100% class purity is rarely achieved in leaf nodes of trees unless the tree is grown to excessive size. There were two considerations in the selection of an appropriately sized tree. The first was that it should not have excessive complexity, and the second closely related point is that a 'sufficient' number of training items should remain in each leaf node to make *k*-NN a viable technique. The idea of what constitutes sufficient items in a leaf is in some ways dependent on the purity of the leaf node and the number *k* being used. Initial thoughts were that the leaf would have to have more than 50 or 100 training items; hence for the size of data sets being considered here (200-800 items) we were only going to consider small trees. However, there is nothing wrong in theory with having as few as two items for a two-class classification decision using NN. As will be discussed later the actual leaf size was not constrained beyond what S-Plus allows (minimum size 5 items). On the complexity issue, since trees generally are pruned to an 'optimal' size that trades off predictive power against complexity (Breiman, et al (1984)), it was intended that the investigation would use classification trees of approximately optimal size. An optimal-sized tree is the one for which cross-validation (defined later in this chapter) has been used to choose the size which minimizes residual deviance.

The use of composite classification rules was first proposed by Darsathy and Sheela (1979). Their approach was to optimally partition the training set feature space so as to allow the deployment of the best classifier in each region. The advantage of this method was that the computationally expensive NN techniques could be forgone in regions where a simple linear discrimination function could be used. The partition was set so that overall accuracy was maintained or enhanced. The current method is similar in that it involves partitioning of the feature space and reduced NN calculations (due to smaller training sets in leaf nodes). It can in fact be called a nested method in that *k*-NN is carried out only in the subsets (leaves) defined by the tree. It is expected that *k*-NN applied inside a leaf node will produce more correct classifications than either the tree alone or *k*-NN using the full training set due to the class purity in the leaves.

2. Algorithm 2. r.d. Abbreviation for Radial Distance.

In this approach, rather than selecting an optimal number *k* of nearest neighbors to consider, we instead consider all the nearest neighbors inside a local radius. The radius is chosen by a search over a range of appropriate radii using leave-one-out cross-validation (defined later in section B.5) in the training set. Once the radius is set, each test item is compared to all the training set members in the local neighborhood thus created. The idea has intuitive appeal similar to the distance weighting and in fact is similar to concepts used in clustering algorithms. This technique does not force any items to any particular space as a clustering algorithm may do; it just allows the neighbors in a dense part of the feature space to provide more information to the classification decision. Where the region is sparse the number of nearest neighbors will be fewer, but it is hoped it will be more accurate than *k*-NN, as nearest neighbors more distant than the optimal distance would have been used in a *k*-NN classification.

O'Callaghan (1975) proposed a composite decision rule in which his *k*-NN rule was augmented by maximum distance and directional constraints. In essence he restricted the distance a neighbor could be from a test item and he did not allow a neighbor to vote if that neighbor was occluded by another in the feature space. The difference in our approach is that the number of nearest neighbors consulted for the classification decision is purely a function of where the test item falls in the feature space. Note the algorithm defaults to 1-NN should the test item have no training set neighbors inside the distance, and in testing the upper limit on NN used was set at 100.

B. TESTING METHODOLOGY

1. Assumptions

The only major assumption made in this thesis is that the efficiency of the methods is not to be considered. That is, the only Measure of Effectiveness for the algorithms is their ability to improve accuracy of the classifications. It is assumed that if the methods have potential for further investigation then efficient and speedy versions can be developed. Hence S-Plus was chosen as the tool and the code in the Appendices is generally very slow.

2. Baseline Classification Methods

Two baseline classification methods were used as the standard for testing. The first was standard *k*-NN with the optimal *k* selected using leave-one-out cross-validation over the training set. Ties in the plurality vote were broken randomly. The second baseline method was a result of the knn-in-leaf method, which produced the need to test against an optimal tree as well. The optimal tree was selected by cross-validation using the trade-off of size versus residual deviance.

3. Data

Data sets were gathered from the UC Irvine Machine Learning Repository (Merz and Murphy (1996)). For simplicity the data sets were selected according to the following criterion:

1. All continuous independent variables with a categorical (class) dependent variable.

2. No missing data.

3. A mid-sized data set, in the range of 200-1000 items.

In addition to the data obtained above one simulated data set based on waveforms (Breiman, et al 1984) was employed.

The data sets which were used in this thesis are described below. The single word name in bold at the start of each paragraph is used throughout this thesis as the name of the data set. Each set is described in sufficient detail to understand the purpose of the classification. Where other results (in terms of achieved misclassification rates) are known they are also given here. In particular a benchmark study of classification techniques, called the Statlog project (Michie, et al 1994), has studied many of the same data sets used in this thesis and their results for the baseline methods are also provided. Further discussion of the Statlog project is given later in this chapter.

Diabetes. Pima Indians Diabetes Database. This data base was developed by National Institute of Diabetes and Digestive and Kidney Diseases and downloaded from the UC Irvine Data Repository. The data contains measurements of 8 attributes from Pima Indian (near Phoenix, Arizona) women aged over 21 and a classification as to whether or not they had diabetes. The data set contained 768 observations of which 500 were class 0, no diabetes. This data set has been well-studied, with the first reported use being by Smith, et al (1988) using their ADAP routine. The Statlog project gave a best misclassification rate of 22.3 % for a Logistic Discriminant routine, whilst CART (classification tree) scored 25.5% and 1-NN had a 32.4% misclassification rate.

Sonar. This is the data set used by Gorman and Sejnowski (1988) in their study of the classification of sonar signals using a neural network. The task was to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The data set consisted of 111 patterns obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions and 97 patterns obtained from rocks under similar conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The data set contains signals obtained from a variety of different aspect angles, spanning 90 degrees for the cylinder and 180 degrees for the rock. Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The best results reported by Gorman and Sejnowski were around 11% misclassification rate for a neural network and 17% for a nearest neighbor classifier. There are no reported results for this data set using classification trees and, as will be seen in Chapter III, for good reason.

Vehicle. Vehicle Silhouettes. The data set was produced by the Turing Institute (Siebert, 1987). The purpose of this data set was to classify a given silhouette as one of four types of vehicle (Saab, Opel, Van, Bus), using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles. The four vehicle types had roughly equal representation in the 846 samples. There were 18 attributes measured. The features were extracted from the silhouettes by the HIPS (Hierarchical Image Processing System) extension BINATTS, which extracts a combination of scale independent features utilizing both classical moments-based measures such as scaled variance, skewness and kurtosis about the major/minor axes and heuristic measures such as hollows, circularity, rectangularity and compactness. Further detail about the data can be obtained at The UC Irvine Data Repository. Statlog results for this data included a best result of 15% for a quadratic discriminant routine, 23.5% for a classification tree and 27.5% for 1-NN.

Image. Image Segmentation data. This data was created by the Vision Group, University of Massachusetts. The set consists of 210 training patterns and 2100 test patterns containing seven classes and 19 measured attributes. Each of the classes is an outdoor image, these being brick-face, sky, foliage, cement, window, path and grass. The Statlog project treated this data set as one large set and did cross-validation on all 2310 items. The results reported thus have a different flavor from those in Chapter III (where the 2100 test items were classified using the 210 training items). The best result for this data set was in a statistical kernel algorithm called ALLOC80 with a misclassification rate of 3%. The classification tree result was 4% whilst 1-NN was 7.7%.

Waves. This data consisted of 300 generated training instances and 3000 generated test instances using an algorithm developed by Breiman, et al (1984) to test classification trees. The set consisted of three classes in roughly equal proportions with 21 attributes randomly drawn from complicated waveform distributions (depending on class) perturbed with random normal noise. The theoretical best possible (Bayes) misclassification rate is 14%. Breiman reported a 28% misclassification rate for the classification tree and 22% (66%) for a NN classifier if the data set was made without (with) the random noise.

4. S-Plus Code and Functions

All algorithms were coded in the S language, in S-Plus version 4.1. The code is built around generic S-Plus functions along with classification functions written by Venables and Ripley (1997). In particular a classification library developed by Venables and Ripley downloaded from Statlib (see Reference List) contained functions to perform *k*-NN and leave-one-out cross-validation *k*-NN.

5. Cross-validation

Two types of cross-validation techniques have been used at various times in this investigation. These are discussed in detail so that the terminology as it is used later is clear. The first is called leave-one-out cross-validation. As the name implies each item in a training set is in turn "left out," or becomes the test item. It is classified against the remaining training set items using the method being examined. Since the left-out item has a known classification the correctness of the classifier can be ascertained. The total number of misclassifications over the training set can be found. This is generally reported as a rate. Note that at various times in this thesis leave-one-out cross-validation is discussed in conjunction with trees. This is not technically correct as the tree is not re-grown for every leave-one-out stage. Instead, the induced tree is applied to all the original data. This could more accurately be called a "self-classification" result. The term leave-one-out is retained as it conveys the flavor of the result.

The second form of cross-validation is the method called n-fold cross-validation. The technique involves randomly splitting the training set into n (usually 10 or 12) subsets or 'chunks'. Each subset is in turn designated as the test set and the remaining n-1 pieces of the set make up the training set. The test set is classified by the method under consideration and the misclassifications accumulated as before as a measure of the accuracy of the method. Finally since there are many ways to randomly split a data set, Monte Carlo sampling (splitting of the data) was done to ensure that any trends seen in the misclassification rate for various methods could be attributed to the method and not just to the vagaries of random subset selection. The number of replications was set to 10, mainly due to time limitations. The n-fold cross-validation technique was used extensively in the Statlog project, so similar n-fold runs were required in this thesis.

6. Common Random Numbers

In the n-fold cross-validation technique, Monte Carlo sampling is required in splitting the data into n subsets. To reduce the variance, and to simplify the comparison of techniques, it was decided to use common random numbers. That is, for the Monte Carlo samples, each method is tested on exactly the same n subsets. This technique is easily implemented in S-Plus as the program has available 1001 preset random seeds for its random number generators. Therefore the seed can be set before any random split of the data was performed.

7. Testing Procedure

The testing procedure was simple. Each data set was classified using the four methods and the code in the appendix. Data sets which consisted of a distinct training set and test set were classified once only. Data sets which were classified using n-fold cross-validation were classified a number of times (arbitrarily set at ten due to time constraints) with different random seeds. The results in the ten n-fold cross-validations were compared using paired *t*-tests for the means, a test applicable because of the use of Common Random Numbers.

In all cases the data sets were also classified using leave-one-out cross-validation as this was considered to be an excellent indicator of the best method to use. Further detailed discussion on the testing and results collected is in the next chapter.

8. Statlog Project

A few comments about the Statlog project are required. This thesis is not testing the new methods against the Statlog methods. The Statlog results are used only to confirm that the observed baseline results are consistent. In order to allow such comparison the methods of this thesis need to be the same as theirs; for example the number of splits for the n-fold cross validation is the same. As will be seen in Chapter III, there are some differences in the observed results. These can be attributed to minor variations in method. A major difference is that Statlog only considered 1-NN whereas we choose an optimal *k* for *k*-NN. In addition, Statlog does not indicate whether they use multiple replications of random n-fold cross-validations or just a single result. If it is a single result then it may be a poor estimator because of the random realization and may not be directly comparable to the results of this thesis. Even if the stated misclassification rate is from an unknown number of replications, the differences observed may well be within variance limits imposed by the random selections which occurred.

III. RESULTS AND DISCUSSION

A. PRELIMINARIES

This chapter will present summary results and discussion for each of the data sets examined in each of four classification methods. (More detailed results are in Appendix B). The four classification methods are the two baseline methods of *k*-NN and classification trees, and the two proposed new methods of knn-in-leaf and r.d. Where other results for the data sets are available from the literature these will be used to see if the methods are "in the ballpark" with regard to overall misclassification rates. All results here will be expressed as misclassification percentages with the raw number misclassified and other data set details such as set size in the appendix. Also in the appendix are details on the optimal *k* used, size of classification trees used, etc.

1. Data Set Types

Two types of data sets were examined in this thesis. One type contained distinct training and test data sets. Waves and Image are data sets of this sort. Members of the second type are single data sets which had to be examined by cross-validation. Thus there will be some small difference in the way the results reported for each should be interpreted. That is, in the case of the distinct training and test data the result is a single answer, which of course might change for different training and test sets. The cross-validation data set misclassification results are observations of random variables. For such cross-validated cases the raw data in the appendix includes the number of Monte Carlo replications and appropriate standard deviations. For each data set examined with Monte Carlo cross-validation the method of Common Random Numbers was used to ensure each method was tested on the same data set splits. Where the new methods show improvement over the baseline methods, the null hypothesis that they have the same population average misclassification rate is tested using one-sided paired *t*-tests. Finally, it should be noted that some data sets respond well to scaling of the variables. By standardizing each variable to have a mean of 0 and a standard deviation of 1 the influences of individual variables are equalized. This is a standard technique in NN classification. Results are also reported for each data set with the data scaled.

The chapter will examine results in terms of misclassification rates as this is the stated primary measure of performance. Other classifier measures of performance including complexity and consistency will be examined later in the chapter.

B. MISCLASSIFICATION RATE RESULTS

Prior to an examination of the results in the table below the differences between the leave-one-out cross-validation and the n-fold cross-validated results needs to be explained. The leave-one-out results are optimistic and could be considered an approximate lower bound on the true misclassification rates. The intention when producing the classification rule should be to use all the available data. Hence 'self testing', as leave-one-out cross-validation could be described, has some benefits, but cannot truly be reported as the potential error rate of the method. The ability of the rule is further tested using n-fold cross-validation, which introduces some errors due to the smaller training sets, but increases our confidence in the ability of the rule as it randomly reorganizes the data. So what can be drawn from this? First, leave-one-out or self - classification results would be an indicator as to which method to use. Then the overall ability of the method, in terms of potential (or expected) misclassification rate, could be reported as the results of the n-fold cross-validation. The main reason that leave-one-out cross-validation is used first is that it requires much less work and is faster. Also, if the result is taken as an approximate lower bound on the ability of the rule in that data set, it enables easy comparison between methods. Finally n-fold cross validation induces n times more work, to get one point estimate of the misclassification rate. To enable comparison of methods more than one replication is required, thus adding more work. Leave-one-out cross-validation does not require multiple replications.

In this thesis leave-one-out cross-validation in methods like knn-in-leaf is an order of magnitude faster than n-fold cross-validation. For this reason a range of parameters for the rule, such as tree size and range of *k* to examine, is investigated fully using leave-one-out cross-validation. Once such data is available a much smaller set of rule parameters need to be examined in the n-fold cross-validation phase.

1. Diabetes Data

Summary results on the misclassification rates for the diabetes data are presented in Table 1. Leave-one-out cross-validation indicates that the knn-in-leaf method should be used and this is validated by the 12-fold cross-validation. The change in rank positions for the classification tree and r.d bears some examination. The misclassifications for the tree are in actuality the self-classification results for the 7 node tree, and it is well known that the misclassification rate is grossly over-optimistic for classification trees. The 12-fold cross-validation result is more indicative of the true ability of the classification tree. Finally it should be noted that the knn-in-leaf results are significantly better than the standard *k*-NN results (p value 0.0002) for the un-scaled data. r.d is also significantly better against *k*-NN with a p value of 0.002.

12-fold Cross-validation		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	25.6%	26.2%	23.5%	24.8%
	Rank	3	4	1	2
Scaled Data	Misclass Rate	24.9%	26.2%	24.3%	26.5%
	Rank	2	3	1	4

Leave-one-out Cross-validation		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	24.0%	22.8%	16.8%	24.1%
	Rank	3	2	1	4
Scaled Data	Misclass Rate	24.0%	22.8%	17.2%	24.3%
	Rank	3	2	1	4

Table 1. Misclassification Rate Results for Diabetes Data

The scaled data results for diabetes supports the general ability of knn-in-leaf, but it is apparent in the leave-one-out results that scaling is not a useful tactic for this data.

The best reported result for this data set in the Statlog project was 22.3% in 12 fold cross validation. The results above do not improve upon this, although the knn-in-leaf leave-one-out cross validation result of 16.8% is impressive. This is discussed later.

2. Vehicle Recognition Data

In this data set, with results which can be seen in Table 2, knn-in-leaf was a significant improvement over the classification tree (p-value 0.025). What is also significant is how scaling the data improved all results, except that scaling does not affect classification trees. A tree produced on scaled data will always be identical to the un-scaled tree. In the scaled data knn-in-leaf was significantly better than the *k*-NN results with a p-value of 0.033. r.d was disappointing; it was on par with *k*-NN in the un-scaled case, but even when scaled was unable to improve upon either the tree or the *k*-NN results.

12-fold Cross-validation		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	35.7%	29.5%	28.3%	35.2%
	Rank	4	2	1	3
Scaled Data	Misclass Rate	28.1%	29.5%	27.2%	29.8%
	Rank	2	3	1	4

Leave-one-out Cross-validation		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	33.2%	23.5%	21.0%	32.9%
	Rank	4	2	1	3
Scaled Data	Misclass Rate	26.7%	23.5%	19.9%	27.5%
	Rank	3	2	1	4

Table 2. Misclassification Rate Results for Vehicle Recognition Data

The best Statlog results for this data was a 15% misclassification rate, using a quadratic discriminant routine. The best result above is not close to that, but these results compare well to the k-NN and classification tree results of that study. (They are not identical for the reasons discussed at the end of chapter II.)

3. Sonar Data

Table 3 contains the summary results for the sonar data set. Whilst it is not indicated in this table, the first thing to note is that the average-sized optimal tree for this data was 2.5 leaves. This is a very small tree and indicates that a classification tree is perhaps not the best tool for this data. This is confirmed when 1-NN is compared to the optimal tree results for this data. It is surprising then that knn-in-leaf is the superior method in leave-one-out cross-validation. This result is not found in the 10-fold cross-validation, but this is more than likely a direct result of the size of the data set and of the number of variables. This set only had 208 items, but 60 variables. It would have been difficult for such a small data set in the n-fold cross-validation to produce trees with consistent break (split) points. This translates into a much higher chance of incorrectly assigning test items to training leaves, and hence a higher knn-in-leaf misclassification rate. It is also further evidence that classification trees do not perform well on this data set. But it is clear that knn-in-leaf was a great improvement over the tree. The r.d result is somewhat misleading for this data as well. In general the optimal distance was so small that the algorithm was defaulting to 1-NN, which obviously is the best k-NN *k* value for this data. Another surprising result given the obvious ability of 1-NN was in the 10-fold cross-validation optimal *k*-NN routine. That is, in some sub-sets the routine chose $k > 1$ as the optimal. This produced worse results than setting $k = 1$ throughout. This could again be attributed to the small data set.

10-fold Cross-validation		1-NN	*k*-NN	Classific-ation Tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	17.5%	19.0%	27.5%	19.4%	17.5%
	Rank	1	2	4	3	1
Scaled Data	Misclass Rate	12.7%	14.2%	27.5%	15.7%	12.7%
	Rank	1	2	4	3	1

Leave-one-out Cross-validation		1-NN	*k*-NN	Classific-ation Tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	17.3%	17.3%	22.6%	13.0%	17.3%
	Rank	2	2	3	1	2
Scaled Data	Misclass Rate	12.5%	12.5%	22.6%	11.1%	12.0%
	Rank	3	3	4	1	2

Table 3. Misclassification Rate Results for Sonar Data

The best result reported for this set was around 11% using a neural network. If the knn-in-leaf leave-one-out result is a reportable result then it is approaching that best result. The best NN classifier result was 17% which is consistent with the results above.

4. Image Segmentation Data

The results for this data set are in Table 4. This data set shows clear superiority of knn-in-leaf (but since this is a single result it cannot be statistically compared), but again a quirk of the data needs to be discussed for r.d. That is, this data responded best in *k*-NN to a *k* of 1. As with Sonar, no improvement to the misclassification rate could be achieved by examining any distance above the one that defaulted the number of nearest neighbors to 1. In contrast to Sonar the optimal *k*-NN routine consistently selected 1-NN for the chunks, which again indicates that the smaller data set may have been a problem in the Sonar case. As will be discussed later in the efficiency of the methods, whilst scaling the data improved the overall misclassification rate for *k*-NN (and r.d by default), the knn-in-leaf results did not significantly change. This could be attributed to the leaves already having a small amount of spatial purity, so that the normalizing effects of scaling were not as severe. That is the splits in the tree had already corrected for some of the effects of scale.

Classification of Test Set		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	12.3%	11.3%	8.7%	12.3%
	Rank	3	2	1	3
Scaled Data	Misclass Rate	9.5%	11.3%	8.7%	9.5%
	Rank	2	3	1	2

Leave-one-out Cross-validation in Training Set		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	15.7%	5.2%	4.8%	15.7%
	Rank	3	2	1	3
Scaled Data	Misclass Rate	12.4%	5.2%	4.3%	14.3%
	Rank	3	2	1	4

Table 4. Misclassification Rate Results for Image Segmentation Data

The results above are a bit higher than the Statlog results for the classification tree and k-NN, as they conducted cross-validation on the combined test and training sets. (These combined data sets are 2310 items which is too large for the code in Appendix A). The best results of a 3% misclassification rate is far superior to the results above.

5. Waves Data

Table 5 shows the test set and cross-validated results for the generated Waves data set. In the test set results r.d shows a small amount of improvement over *k*-NN. Scaling has only minor effects on the data, but this is to be expected due to the way the data was generated. That is, each variable had the same possible range, and in effect the data was already scaled. But what did happen in the scaled results was that *k*-NN was superior to r.d. The classification tree responded very poorly to the data, and knn-in-leaf was able to improve upon that, but not significantly when compared to *k*-NN. Knn-in-leaf did not vary between scaled and un-scaled for similar reasons as in the Image data.

Classification of Test Set		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	17.5%	37.4%	22.1%	17.0%
	Rank	2	4	3	1
Scaled Data	Misclass Rate	16.7 %	37.4%	22.4%	17.5%
	Rank	1	4	3	2

Leave-one-out Cross-validation in Training Set		*k*-NN	Classification tree	knn-in-leaf	r.d
Un-Scaled Data	Misclass Rate	15.0%	28.0%	14.7%	13.3%
	Rank	3	4	2	1
Scaled Data	Misclass Rate	17.0%	28.0%	14.7%	15.0%
	Rank	3	4	1	2

Table 5. Misclassification Rate Results for Waves Data

This data set has a theoretical (Bayes) best possible misclassification rate of 14%. The results for the classification tree are consistent with those previously reported by Breiman, et al (1982). The k-NN results are much better than Breiman's results, but it is likely that result was from 1-NN.

C. OTHER CLASSIFIER MEASURES OF EFFECTIVENESS

1. Magnitudes of Improvement

In all data sets knn-in-leaf was able to show some improvement over the classification tree. In three out of five data sets knn-in-leaf was the best of all. In the Sonar data knn-in-leaf appears to have failed due to the fact that the tree responds very poorly to the data set. In the last set (Waves) r.d was better (but this is examined more closely later). When examining the leave-one-out cross-validation results, the knn-in-leaf was superior in nine out of 10 cases. But what needs to be considered is whether or not the superiority is worth the extra effort. In reality this has to be a function of the cost of the misclassification. When a misclassification has a high cost even a small improvement will be valuable. Since there is no data about the cost of misclassification, and in reality we are unable to accurately quantify the extra effort required anyway, a basic measuring stick approach should provide some guide.

In most of those cases where knn-in-leaf was better it improved by more than 1% compared to the next best classifier. The improvement was smaller for scaled data, perhaps because the tree had already corrected for scale. It is appropriate at this point to discuss how the tree corrects for scale. The scale correction is not to the same extent as coercing all variables to have mean 0 and standard deviation 1, but if one variable did have large fluctuations compared to other variables, a split in its middle could have large impact. For example, consider the case where most variables had range 0-1, but one was 0-10. Some neighbors which may be essentially equal in all other ways but with opposite extreme values of the latter variable will already be more than 10 units apart by virtue of being at either extreme of this variable. If a split occurred on this variable around the middle of its range, then the first thing to note is that those two instances can no longer be neighbors since they are in different leaves. Also observations at the extremes – 0 to5 or 5 to10 in respective leaves - only start around 5 units apart before other variables are considered. The point is such extreme distance effects may be reduced in the leaves. This is the scaling effect.

In the Waves data where r.d was ranked first it was only 0.5% better than *k*-NN, which was reversed to 0.8% behind *k*-NN in the scaled data. The greatest magnitude of knn-in-leaf improvement was 2.6% in the Image data over the classification tree, which reduced to 0.8% over *k*-NN in the scaled case. Diabetes showed similar improvement with 2.1% over *k*-NN reducing to 0.6% in the scaled case. Thus if a 1-2 % improvement is warranted the knn-in-leaf method should be considered.

2. Work and Complexity

These terms are used loosely in the following discussion about the size of the classification tree and relative number of nearest neighbors to examine. It should be noted that the efficiency of any of the algorithms in this thesis has not been seriously considered. In many cases all of the distances from test items to training items have to be calculated at some stage, and we assume these can be stored easily since memory is cheaply available. Thus in a comparison of the relative merits of ten nearest neighbors versus 50 nearest neighbors, the relative speed of lookup is really not an issue. What is more important is the conceptual leap. Is it better to only have to consider ten NN or 50? There are also complexity issues in a tree and these are discussed below as well.

Table 6 below contains summary information about the size of classification trees and number of nearest neighbors examined for each data set and method. Disregarding the cases where $k = 1$ was optimal causing r.d to default to a small distance, the first point to note is a general trend which is immediately apparent for r.d. This is, it consistently examined many more nearest neighbors than the other methods. Conversely knn-in-leaf consistently used no more nearest neighbors than *k*-NN. An interesting aside is that when $k = 1$ was optimal for the data set as a whole in *k*-NN, sometimes knn-in-leaf actually used $k > 1$ nearest neighbors and was able to improve the misclassification results. (Note the *k* used in knn-in-leaf is an average over all the leaves that does not take account of the number of items in a leaf or number of leaves, which in some regards is misleading. For example a tree of 5 leaves may have had an optimal *k* of 1 in four of them and a *k* of 6 in the 5th, leading to an average *k* of 2.)

	Diabetes	Vehicle	Sonar	Image	Waves
Un-Scaled Data					
k used in *k*-NN	19	3	1	1	15
k used in knn-in-leaf	9	3	4	2	7
k used r.d	76	6	1	1	32
Tree size (leaf nodes) Classification tree	7	16	3	10	5
Tree size (leaf nodes) knn-in-leaf	9	17	3	10	4
Scaled Data					
k used in *k*-NN	22	5	1	1	21
k used in knn-in-leaf	5	3	3	2	11
k used r.d	60	15	1	1	56
Tree size (leaf nodes) Classification tree	7	16	3	10	5
Tree size (leaf nodes) knn-in-leaf	9	17	3	10	3

Table 6. Comparison of Classifier Complexity and Work

The second point is in regard to the optimal size of the knn-in-leaf tree versus the classification tree. The optimal size classification tree is chosen by a trade-off between size and predictive ability as measured by residual deviance. In general the optimal knn-in-leaf tree was similarly sized. Size was chosen by cross-validating over a range of tree sizes and looking for the minimum misclassification rate. Three distinct cases need to be examined here. The first was in Diabetes. In the leave-one-out results for knn-in-leaf there was a significant improvement of the 9 leaf tree over any other method, and over knn-in-leaf of any smaller tree. This may have been just luck, but there was real improvement in the predictive ability of the method. The 8 leaf knn-in-leaf tree had 159 misclassifications, the 9 leaf tree had 129 and the 10 node tree, 141. The improvement from the 8 leaf tree to the 9 leaf was 30 misclassifications. Put another way, 30/768 was a raw 4% improvement, but what makes this impressive was the comparison to the classification tree of the same size. The tree, which we know self-classifies better when the size is greater, misclassified 175 for an 8 leaf tree, which then improved to 161 for the 9 and 10 leaf tree. The knn-in-leaf result was an improvement of 32 over the 9 leaf tree. Even though this event occurred in the leave-one-out cross-validation which is the optimistic lower-bound end of the scale, this is an impressive result.

The second special case occurs for the Image Data. In the leave-one-out cross-validation results for the classification tree and knn-in-leaf, it was noted that as the tree got larger and larger the knn-in-leaf method continually got better. That is, there was no detectable minimum. But at the same time it was clear that the relative difference between the tree and *k*-NN generally remained the same. There were two possible approaches to choosing the optimal sized tree for knn-in-leaf. The first was to look for an obvious "big jump" change in the relative misclassifications; the second was to apply rules that already existed for the classification tree. That is, an optimal sized classification tree would be selected, so that tree may as well be used for the knn-in-leaf. In this data set the second method was used.

The third case is an examination of the effectiveness of large trees. In the Vehicle data, the optimal tree had 16 leaf nodes. The large tree can be deceptive, as excellent results are usually achieved in self classification. As discussed above n-fold cross-validation is the method used to confirm the predictive ability of the large tree. As can also be seen for this data the optimal knn-in-leaf tree has 17 leaf nodes. The question has to be asked, can knn-in-leaf be effective in such a large tree? Large trees imply smaller leaf nodes, which may reduce the ability of *k*-NN. Also any tree misallocate some portion of test items to leaves, these being the misclassification errors. It appears from the results, that even in a large tree, knn-in-leaf may be able to reduce some of those errors. That is, a test item may be allocated to the a leaf of the wrong class label, but *k*-NN uses all training items in that leaf, allowing some otherwise incorrect classifications to be corrected. Also, it appears that those small leaves have a level of purity that allows a smaller *k* to be used, and that also allows correct classification of other items in the leaf.

So in consideration of work and complexity, any knn-in-leaf tree which is similarly sized to the optimal tree has similar initial complexity. Further work is added in conducting *k*-NN in leaves. But, in comparison to *k*-NN and r.d, significantly fewer nearest neighbors have to be considered. Along with that reduction in work, since the training set in the leaf is smaller, significantly fewer distance calculations are required. The r.d method required a larger number of nearest neighbors to be consulted, but since we have already assumed that memory is cheap, such a comparison may well be easily accomplished once the computation is completed.

3. Consistency of Classifier

The final measure of effectiveness for a classifier is that its results are consistent. There are two ways that such a judgement will be made in this thesis. The first is whether the leave-one-out cross-validation results were confirmed by the n-fold cross-validation or the test set classification as applicable. It is clear that knn-in-leaf was the most accurate method for most of these data sets in the leave-one-out cross-validation (nine out of 10 cases). The method was consistent in the follow up n-fold cross-validation or test set results in six out of the nine cases. Two of the inconsistent cases were the scaled and un-scaled Sonar data which appears to have suffered from problems, perhaps due to the small sample size. This leaves one case, that of the scaled Waves data, as the inconsistent case (in that the test set knn-in-leaf was not the best as indicated by the leave-one-out cross-validation results). But closer examination of the Waves data shows that the difference between knn-in-leaf and r.d in leave-one-out cross-validation was only 0.3% or 1 correct classification. So it is not clear that knn-in-leaf was obviously best for that case.

In the only case where r.d was the superior method, it was also the superior method in the test set classification.

The second consistency judgement is made only in the case of n-fold cross-validation. That was the amount of variance observed among misclassification rates in successive Monte Carlo samples. In general the sample standard deviation of knn-in-leaf misclassifications was on par with that of the classification tree, but usually higher than the *k*-NN and r.d results. Since the results were on par with the classification tree, they are at least as consistent as the tree.

IV. CONCLUSIONS AND FURTHER RESEARCH

As always when working with real data there is never one best way to analyze it, or in this case classify it. It is apparent that *k*-NN is a viable first approach to many classification problems, but as has also been seen in this thesis, some form of preprocessing or embellishment may improve the ability of the classification rule. Of the two methods examined in this thesis, knn-in-leaf has some merits and these are discussed below. The second, r.d, was disappointing and again indicated how intuition can often fail in practice. As a final summary, the ranking frequency for each method over the scaled and un-scaled data sets is presented below in Table 7.

n-fold Cross-validation or Test Set Classification			
Frequency	**Ranked 1**	**Ranked 2**	**Ranking > 2**
Knn-in-leaf	6	2	2
k-NN (or 1-NN)	3	4	3
Classification tree	0	2	8
r.d	1	5	4
Leave-one-out Cross-validation			
Knn-in-leaf	9	1	0
k-NN (or 1-NN)	0	1	9
Classification tree	0	6	4
r.d	1	3	6

Table 7. Summary of Rankings of Each Classification Method

A. KNN-IN-LEAF

1. Conclusions

It is clear from the results that knn-in-leaf has merit for further investigation. The leave-one-out cross-validation results showed that the method was able to improve the approximate lower bound (leave-one-out cross-validation) on the misclassification rates for nearly all the data sets examined. It was better than the baseline methods in all cases. In the n-fold and test set classification results, the indicative results were confirmed in six out of the nine cases. As was discussed in chapter III, at least two of the cases where it failed to make an impact could have been due to limited size of the data set. The overall impact of the method in percentage terms was not large, a 1-3% improvement in the misclassification rate in most cases; but if the cost of misclassifications was high such an improvement would be useful.

2. Further Research

The knn-in-leaf method grew out of an attempt to improve classifications by locating areas of class purity in the feature space. The classification tree was chosen as a simple splitting tool. It is clear that classification trees may not be the best way to find regions of class purity. Clustering algorithms provide another way of doing such feature space splitting. Such algorithms were not tried in this thesis as they usually already use distance-based approaches, and it is not clear that we would have been able to exploit different information available in the data. Thus, one further direction for research would be alternative ways of finding areas of class purity in the feature space. These may take the form of existing clustering techniques or some completely novel way of assigning training set items to regions of class purity. For example, splits could be chosen in an explicit search over some or all variables seeking to actually minimize the final misclassification rate.

The other direction of further research should remain in the classification tree. In early work on the Diabetes data it was clear that the improvements in misclassifications were occurring in non-pure leaves. That is, where a leaf was already 80% or more pure in a class type, there was little if any improvement when *k*-NN was conducted. In fact both methods were misclassifying the same items. This effect should be investigated for other data sets as it leads to two observations:

1. If we can identify that a leaf is (class) pure, then there is no need to do knn-in-leaf in that leaf. Of course the work here would be to define what a pure leaf was, and then in fact to decide whether to save work and not do knn-in-leaf, or maybe to do a quick 1-NN check only.

2. The second is that we have the opposite effect to that sought when this thesis was started. That is, we sought regions of class purity to improve the misclassification rate, but actually didn't improve it in the regions of class purity at all. Rather the improvement in misclassification rates have occurred in the remaining areas of impurity. Thus, perhaps the focus in the search for a splitting rule should be on separating out the regions of the feature space where misclassifications often occur and seeking a new classification rule there. This is in some regards the composite approach of Dasarathy (1979). Further research could look to those items that are often incorrectly classified (by different classifiers), to see when and if they are ever correct, and if so, to try and draw the information that causes such a correct classification into the splitting decisions. This may remain some form of nested method.

The final areas of research for knn-in-leaf relate just to the way it is currently implemented in a classification tree. These are detailed below:

1. There needs to be an examination into whether there is a size of leaf node that would make *k*-NN non-viable. Because a smaller k is used in leaves than in the usual k-NN technique, small leaves may well be viable, but the question is how small is small? As discussed above the leaf purity is also an issue. The aim would be to produce work-reducing rules to be implemented in the code.

2. Examining the currently coded algorithms it is clear that there is a lot of wasted effort in doing cross-validations over multiple *k* levels in small leaves. These algorithms can be made smarter.

3. Scaling has been shown to improve *k*-NN misclassification rates in most cases. But scaling has no effect on trees as the splits work out the same. Since *k*-NN benefits from scaling it is probable that scaling within leaves could improve the misclassification rates. This could easily be added to the current S-Plus code.

B. R.D

1. Conclusion

r.d was unable to appreciably improve the quality of any classification decision over *k*-NN. At the same time the conceptual work being done was significantly higher as it consistently examined a higher average number of nearest neighbors. It is not recommended that r.d receive any further research effort at this time.

APPENDIX A. S-PLUS CODE

This appendix contains the S-Plus code for functions used to test the classification methods in this thesis. The heading for each function is its name, and the first comment block provides a description of the purpose of the function. Note that some names are a bit unclear, but since functions are called from within other functions the names have not been changed here.

Other functions used included standard S-Plus functions for classification trees (S-Plus, 1997), along with a library of classification functions produced by Venables and Ripley (1994). These functions are also described below.

A. *k*-NN FUNCTIONS

1. knn

This function is a standard *k*-NN algorithm for classifying a test set against a known training set. It was part of a library of classification functions written by Venables and Ripley. The function returns the classification of the test set instances. This function is heavily used within other functions.

2. knn.cv

This function is another from the Venables and Ripley classification library. It performs leave-one-out cross-validation on a training set. The function returns the predicted classifications of the training set. It is also heavily used in other functions.

3. daisy

This is an S-Plus function, from the cluster library, that takes in a data frame or matrix of data and returns a vector of Euclidean distances between all pairs of rows of the matrix. This is the basis of any NN algorithm; to find out how close each item is to all others.

4. knn.cross.val

```
function(data, classes, n = 10, kstep = 1, no.k = 8, verbose = F)
{
#
# By Sam Buttrey and Ciril Karo
#
# knn.cross.val: Do n-fold k-NN cross-validation on the full data
# set.  Randomly split the data into n chunks.   Loop over each
# chunk in turn as test set, remaining is training set.  Search
# for optimal k in training chunk using leave one out cross
# validation (knn.cv).  Conduct knn for test chunk and sum up
# misclassifications
#
# Arguments:
#  data     :  full set of data to be used in cross validation
#              (without classes)
#  classes  :  vector of classes of the full set of data
#  n        :  number of chunks to split the data into
#              (default, 10) (They won't necessarily be exactly
#              the same size.)
#  kstep    :  1 or 2 depending on what is known about the likely
#              range of k for the data set.  The function searchs
#              over the range of k defined by this step and no.k
#              below, seeking the minimum number of
#              misclassifications
#  no.k     :  in conjuntion with kstep, works out how many k's to
#              examine in the cross validation of the training set
#  verbose : whether or not to dump extra information to screen
#
# return value   :     sum of misclassifications
#                :     average optimal k selected in each cross
#                      validation chunk
#
# Get the sample. Set up the vector of places where the chunks
# start as a vector of evenly-spaced non-integers; then round
# them. The last one is exactly nrow(data); that's okay.
#
   samp <- sample(1:nrow(data))
   chunk.start <- round(seq(1, nrow(data), len = n + 1))
         [ - (n + 1)]  #
#
# Loop through the chunks. Each chunk goes from its starting
# point to one less than the next starting point (except that the
# last chunk includes the final data point.) For each chunk, find
# optimal k to use, using leave one out cv the use Ripleys knn on
# test chunk and get the misclassifications
#
   Sum.Misclass <- 0
   tempbestk <- numeric(no.k)
   opt.k <- 0
   for(i in 1:n) {
      if(i == n)
         out <- samp[(chunk.start[i]):nrow(data)]
      else out <- samp[(chunk.start[i]):(chunk.start[i + 1] - 1)]
#
# This loop finds the misclass sum for each k, stores in
# tempbest[], order will get back the location of the k to use,
# and now depending on what k step was you know what k to use in
# knn() for the test chunk
#
      for(k in 1:no.k) {
         if(kstep > 1)
            tempbestk[k] <- sum(classes[ - out] != knn.cv(
                  data[ - out,  ], classes[ -out], kstep * k - 1))
         else tempbestk[k] <- sum(classes[ - out] != knn.cv(
               data[ - out,  ], classes[ - out], k))
      }
      bestk <- order(tempbestk)[1]
      if(kstep > 1)
         this.knn <- knn(data[ - out,  ], data[out,  ],
               classes[ - out], kstep * bestk - 1)
      else this.knn <- knn(data[ - out,  ], data[out,  ],
            classes[ - out], bestk)
      Sum.Misclass <- Sum.Misclass + sum(this.knn !=
            classes[out])
      if(verbose) {
         if(kstep > 1)
            cat("end chunk ", i, " opt k ", kstep * bestk - 1,
                  " Misclass to date ", Sum.Misclass, "\n")
         else cat("end chunk ", i, " opt k ", bestk,
               " Misclass to date ", Sum.Misclass, "\n")
      }
      opt.k <- opt.k + bestk
   }
   if(kstep > 1)
      avg.opt.k <- kstep * (opt.k/n) - 1
   else avg.opt.k <- opt.k/n
      return(list = c(Misclass = Sum.Misclass, Avg.Opt.k =
            avg.opt.k))
}
```

5. new.sams.knn

```
function(trng, test, trng.classes, k, trng.chunk = min(300,
nrow(trng)), test.chunk = min(200, nrow(test))) {
#
# By Sam Buttrey and Ciril Karo
#
# new.sams.knn: Even more excellent knn function.  Knn from
# Ripley's library only returns classes, but we need distances
# and other information.   This function will produce that
# information.  Since we need to be able to handle any sized
# data sets, and Daisy() is limited in its speed for sets bigger
# than 500 by 500, there is some crazy-looking data manipulation
# inside this function.  This function is not fast or efficient,
# it exists to produce data to enable us to prove a concept.
#
# Args: trng: training set
#       test: test set
#    classes: Classes of training set
#          k: Number of nn's
# trng.chunk: Number of training rows to consider per iteration
# test.chunk: Number of test rows to consider per iteration
#
# Return value: list of three matrices, each nrow(test) x k:
# One contains indices of the nn's, one contains their classes
# and the last, distances to the relevant test item.
#
# Check to see "daisy" exists. If not, get it. Also, if "test" is
# a vector, make it a matrix.
#
  if(!exists("daisy")) library(cluster)
  if(!is.matrix(test)) test <- matrix(test, nrow = 1) #
#
# Create two of the result matrices. The last gets made at the
# end.  Start them off with negative distances.
#
  indices <- dists <- matrix(-1, nrow(test), k)  #
#
# Set up starting points for loops.
#
  first.test.row <- first.trng.row <- 1
  last.test.row <- test.chunk
  last.trng.row <- trng.chunk    #
#
# Big outer loop: do test set piece by piece
#
  while(1) {
    nrow.this.test.chunk <- last.test.row - first.test.row + 1#
#
# Big inner loop: do trng set piece by piece.
#
    while(1) {  #
#
# Create one big data set out of test and training data; then
# get the distances.
#
      big.data <- rbind(test[first.test.row:last.test.row,  ],
           trng[first.trng.row:last.trng.row,  ])
      nrow.this.trng.chunk <- last.trng.row - first.trng.row + 1
      all.dists <- daisy(big.data)  #
```

```
#
# Okay. Daisy gives us a weird vector in which there is one
# distance from each row to the ones farther down in the data
# frame. So the distances for row 1 extend from 1 to (n-1), where
# n = nrow(big.data); the distances for row i extend from (end of
# those for (i-1)) to (end of those for(i-1), plus n, minus i).
# The ones we want are the last "chunk" of those; those
# are the distances to members of this training set chunk.
#
        nrow.big.data <- nrow(big.data)
        for(i in 1:nrow.this.test.chunk) {
           if(i == 1) {
             start <- 1
             finish <- nrow.big.data - 1
           }
           else {
             start <- finish + 1
             finish <- finish + nrow.big.data - i
           }
           its.dists <- all.dists[(finish -
                nrow.this.trng.chunk + 1):finish]      #
#
# Its.dists are the relevant ones. We need to see if any of these
# dists are smaller than the ones already in place (except that,
# first time through, everything is negative, so that's a special
# case). The "ordering" comes out of "order," except we need to
# add (first.trng.row - 1) so that we refer to rows in the whole
# training set, not just in this chunk. There's no need to keep
# track of the classes yet; we can do that en masse at the end.
# "Dists.row" is the number of the result row, or, equivalently,
# the number of the current test row (whereas i is the number
# just within the current chunk).
#
           dists.sorted <- sort(its.dists)
           dists.order <- order(its.dists) + first.trng.row - 1
           dists.row <- i + first.test.row - 1
           if(any(dists[dists.row,  ] < 0)) {
             dists[dists.row,  ] <- dists.sorted[1:k]
             indices[dists.row,  ] <- dists.order[1:k]
           }
           else {
#
# Otherwise, it's not so easy. We need to compare the newly-
# computed distances with those we had before. They all started
# negative, so if we see any negative distances, we jam in
# whatever we've got.
#
             if(any(its.dists < max(dists[dists.row,  ]))) {
               big.dists <- c(dists[dists.row,  ], its.dists)#
#
# "Big.dists" contains k previous distances (that is, the k
# smallest we've encountered so far) plus all current distances.
#
                 big.inds <- c(indices[dists.row,  ],
                        (first.trng.row:last.trng.row))
                 overall.order <- order(big.dists)[1:k]
                 dists[dists.row,  ] <- big.dists[overall.order]
                 indices[dists.row,  ] <- big.inds[overall.order]
               }
             }
          }
# end "for" loop on test chunk.
#
# Having updated the results, it's time to move on to the next
# chunk of the training set, if there is one. If not, it's time
# to move to the next test chunk.
#
          if(last.trng.row < nrow(trng)) {
             first.trng.row <- last.trng.row + 1
             last.trng.row <- last.trng.row +
                  min(trng.chunk, nrow(trng) - last.trng.row)
          }
          else break
       }
# end "while" on chunks of training data; below is the end for
# chunks of test data.  Whenever we change test chunks we need to
# return to training chunk #1.
#
       if(last.test.row < nrow(test)) {
          first.test.row <- last.test.row + 1
          last.test.row <- last.test.row +
               min(test.chunk, nrow(test) - last.test.row)
          first.trng.row <- 1
          last.trng.row <- trng.chunk
       }
       else break
    }
#
# Finally, we go back and get the classes.
#
   classes <- matrix(trng.classes[indices], nrow(indices),
        ncol(indices))
   return(list(Dists = dists, Indices = indices,
        Classes = classes))
}
```

6. majority.big

```
function(classes, k)
{
# By Sam Buttrey and Ciril Karo
#
# majority.big: Big ol' nn voter.  Give the class of a row of
# classes matrix produced by new.sams.knn() for a given k
#
#  Inputs - classes: Matrix of classes, nrow(test) x (some number
#                    of nn)
#                 k: Vector of nn's to consider for plurality
#                    vote, one k for each row of the test set.
#
#  Output - nrows(test) vector of estimated classes of each test
#           item
#
# Here's a function that takes a vector of classes, plus a k. It
# builds a table of the first k classes and finds the largest
# one, breaking ties at random if necessary.
#
   ruler <- function(x, k)
   {
      get("i", frame = 1)
      y <- table(x[1:k[i]])
      winner <- as.numeric(names(y)[y == max(y)])
      assign("i", i + 1, frame = 1)
      if(length(winner) > 1)
         return(winner[sample(1:length(winner), size = 1)])
      else return(winner)
   }
#
# Cool. Now apply that function to each row of "classes" and
# split.
#
   assign("i", 1, frame = 1)
   return(apply(classes, 1, ruler, k))
}
```

B. knn-in-leaf FUNCTIONS

1. knn.tree.cv

```
function(data, classes, n = 10, treesize = 4, kstep = 1,
no.k = 8, seed = 0, opt.tree = T, verbose = F)
{
#
# By Ciril Karo
# knn.tree.cv: do n-fold cross-validation on the full data set in
# tree using knn-in-leaf and at the same time cross validate the
# optimal tree.  The optimal tree is chosen by cv.tree(), an
# internal S-Plus function that gives the ability to trade off
# tree size to residual deviance.  In addition if n=1 then leave
# one out cv is to be conducted
#
# Arguments:  data : full of data to be used in cross validation
#                    (without classes)
#          classes : vector of classes of the full set of data
#                    (assume numeric classes in form  0,1,2 etc)
#          n       : number of chunks to split the data into
#                    (default, 10). (They won't necessarily be
#                    exactly the same size.).  If n=1 the case is
#                    that leave one out cross validation is to be
#                    conducted
#          treesize: scalar size of knn-in-leaf tree to cv
#          kstep   : steps size for search for optimal k in leaf
#          no.k    : number of ksteps to search for opt k in leaf
#          seed    : used when want to examine various knn-in-
#                    leaf trees using Common Random Numbers.
#                    Note:  seed=0 bypasses the seed setting
#          opt.tree: whether or not to calculate optimal tree
#                    (if doing multiple runs can save work)
#          verbose : whether optimal tree size and other
#                    information is to be dumped to screen
#
#   return value : sum of misclassifications knn-in-leaf
#                : sum of misclassifications in optimal tree
#
# The overall Plan for this function to do cross validation on
# the knn-in-leaf method is as follows:
# Take in data set - split into chunks, looping over each chunk
# in turn build a tree of treesize nodes using data less the
# current test chunk, assign elements of test chunk to a leaf.
# Do knn.cv (leave one out cv) inside leaf to find optimal k,
# then do knn on the test chunk pieces in the leaf, count up
# misclassifications.
# Meanwhile built the optimal sized tree, and predict with the
# test chunk in the tree as well, counting misclassifications.
# goto next chunk and do it all again, report final
# misclassification sums.
#
# Additionally if n=1, then don't chunk, just do leave one out cv
# inside leafs of tree.  That is build tree of treesize, inside
# each leaf do leave one out cv and count misclassifications.
#
# 1. Get the sample. Set up the vector of places where the chunks
# start as a vector of evenly-spaced non-integers; then round
# 'em. The last one is exactly nrow(data); that's okay.  Only
# required if n>1.
#
   if(n > 1) {
      if(seed != 0)
         set.seed(seed)
      samp <- sample(1:nrow(data))
      chunk.start <- round(seq(1, nrow(data), len = n + 1))
           [ - (n + 1)]
   }
#
# Loop through the chunks. Each chunk goes from its starting
# point to one less than the next starting point (except that the
# last chunk includes the final data point.) For each chunk, do
# stuff listed above.
# first of all some set up variables and arrays
#
   Sum.Misclass <- Tree.Sum.Misclass <- 0
        avg.opt.treesize <- avg.opt.k <- 0
   opt.k <- numeric(treesize)
   temp.cv <- numeric(no.k)
   big.data <- cbind(class = classes, data)
   big.data <- data.frame(big.data)  #
#
# this is cheating a bit, assumes that classes are numeric 0,1,2
# etc.   Maybe I'll fix later - nope!
#
   numClass <- max(classes) + 1
   assign("numClass", numClass, frame = 1)  #
#
# Now big if check to see what case I am doing - n-fold or leave
# one out CV
#
   if(n != 1) {
# Now we are in n-fold CV, and need to loop over n chunks
      for(i in 1:n) {
         if(i == n) out <- samp[(chunk.start[i]):nrow(data)]
         else out <- samp[(chunk.start[i]):
               (chunk.start[i + 1] - 1)]  #
#
# build the test and training sets and build/prune tree
#
```

```
        temp.data <- big.data[ - out,  ]
        tst.data <- big.data[out,  ]
        assign("tst.data", tst.data, frame = 1)
        assign("temp.data", temp.data, frame = 1)
        big <- tree(as.factor(class) ~ ., data = temp.data)
        assign("big", big, frame = 1)
        small <- prune.tree(big, best = treesize)  #
#
# find optimal-sized small tree.  Seed required if want same tree
# selected each time.  If we are in a common random numbers
# situation we want the same tree for each chunk (in each of the
# CRN reps) so this seed + i setting fixes that.
#
        if(opt.tree) {
           if(seed != 0)
             set.seed(seed + i)
           out <- cv.tree(big, FUN = prune.tree)
           opt.treesize <- out$size[order(out$dev)[1]]
#
# since trees of size 1 do not make sense just default to 2.  An
# alternate approach would be to take the second lowest deviance
# which could be added later
# eg code required: opt.treesize<-out$size[order(out$dev)[2]]
#
           if(opt.treesize == 1) {
             opt.treesize <- 2
             cat(" opt tree size changed from 1 to 2")
           }
           if(verbose) {
             cat("optimal tree size chunk  ", i,
                 " is ", opt.treesize, "\n")
           }
           avg.opt.treesize <- avg.opt.treesize + opt.treesize
           opt.small <- prune.tree(big, best = opt.treesize)#
#
# predict the current optimal small tree, so as to do comparison
# of the tree method versus the knn-in-leafs.
#
           dpred <- predict(opt.small, newdata = tst.data[, -1])
           dpred <- apply(dpred, 1, function(x)
                 (0:(numClass - 1))[x == max(x)])#
#
# Handle ugly case where there are ties (and therefore dpred is a
# list)
#
           if(is.list(dpred)) {
             dpred <- sapply(dpred, function(x)
             {
               if(length(x) > 1)
                 x[sample(1:length(x), size = 1)]
               else x
```

```
                }
                )
              }
              Tree.Sum.Misclass <- Tree.Sum.Misclass +
                   sum(dpred != tst.data[, 1])
            }
# end if (opt.tree)
#
# from small I want the leaf node numbers.  This ugliness will
# get them for me
#
            leafs <- as.numeric(dimnames(small$frame)[[1]]
                 [small$frame[, "var"] == "<leaf>"])    #
#
# now predict which leaf each chunk element will be in using
# predict.leaf written by Sam Buttrey
#
            chunk.leaf <- predict.leaf(small, tst.data[, -1])    #
#
# time to loop over each leaf in turn to do knn-in-leaf
#
            for(j in 1:treesize) {
#
# identify the training and test elements in the leaf
#
              sub.leaf <- temp.data[identify(small, leafs[j]),  ]
              tst.leaf <- tst.data[chunk.leaf == leafs[j],  ,
                   drop = F]
              if(verbose) {
                cat("trg.leaf has ", nrow(sub.leaf), " items \n")
                cat("test.leaf has ", nrow(tst.leaf), " items \n")
              }
              if(nrow(tst.leaf) == 0) next    #
#
# now do cross validation in each leaf to determine optimal k
# note this is Ripley's leave one out cv function.
#
              for(k in 1:no.k) {
                if(kstep > 1) {
                  temp.cv[k] <- sum(sub.leaf[, 1] !=
                       knn.cv(sub.leaf[, -1], sub.leaf[, 1 ],
                       kstep * k - 1))
                }
                else {
                  temp.cv[k] <- sum(sub.leaf[, 1] !=
                       knn.cv(sub.leaf[, -1], sub.leaf[, 1 ], k))
                }
              }
#  end for loop over number of k steps
              opt.k <- order(temp.cv)[1]
              avg.opt.k <- avg.opt.k + opt.k
```

```
          if(verbose) {
            if(opt.k == no.k)
              cat(" opt.k = no.k,
                   so you may want to increase no.k \n")
          }
#
# use Optimal k to get misclass for test sample and add to
# misclass sum.
#
          if(kstep > 1) {
            Sum.Misclass <- Sum.Misclass +
               sum(knn(sub.leaf[, -1], tst.leaf[, -1, drop = F],
               sub.leaf[, 1], kstep * opt.k - 1) !=
               tst.leaf[, 1])
          }
          else {
            Sum.Misclass <- Sum.Misclass +
               sum(knn(sub.leaf[, -1], tst.leaf[, -1, drop = F],
               sub.leaf[, 1], opt.k) != tst.leaf[, 1])
          }
        }
# this brace is closing loop over leafs
#
# end if n != 1 loop, now loop to next chunk and do it all again
  }
#
# if n=1 we are doing leave one out cross validation, much
# simpler but just messy code
#
  if(n == 1) {
# build the tree
      big <- tree(as.factor(class) ~ ., data = big.data)
      assign("big", big, frame = 1)
      assign("big.data", big.data, frame = 1)
      small <- prune.tree(big, best = treesize) #
#
# dump summary() to screen so I can read off the tree self
# classification misclassifications
      if(verbose) {
         cat("summary of current tree", summary(small), "\n")
      }
#
# from small I want the leaf node numbers.
      leafs <- as.numeric(dimnames(small$frame)[[1]]
            [small$frame[, "var"] == "<leaf>"])    #
#
# loop now over each leaf in the tree
      for(j in 1:treesize) {
#
# identify the elements in the leaf
#
```

```
        sub.leaf <- big.data[identify(small, leafs[j]),  ]
        if(verbose) {
           cat("trg.leaf has ", nrow(sub.leaf), " items \n")
        }
        if(nrow(sub.leaf) == 0) next   #
#
# now do cross validation in each leaf to determine optimal k
# note this is Ripleys leave one out cv function
#
        for(k in 1:no.k) {
           if(kstep > 1) {
             temp.cv[k] <- sum(sub.leaf[, 1] !=
                 knn.cv(sub.leaf[, -1], sub.leaf[, 1],
                 kstep * k - 1))
           }
           else {
             temp.cv[k] <- sum(sub.leaf[, 1] !=
                 knn.cv(sub.leaf[, -1], sub.leaf[, 1], k))
           }
        }
        opt.k <- order(temp.cv)[1]
        avg.opt.k <- avg.opt.k + opt.k
        if(verbose) {
           if(kstep > 1) {
             cat("opt k for leaf ", leafs[j], " is ",
                 opt.k * kstep - 1, "\n")
           }
           else {
             cat("opt k for leaf ", leafs[j], " is ",
                 opt.k, "\n")
           }
        }
        Sum.Misclass <- Sum.Misclass + sort(temp.cv)[1]
        if(verbose) {
           if(opt.k == no.k)
             cat(" opt.k = no.k, so you may want to
                 increase no.k \n")
        }
# end leaf of tree loop
     }
#
# now predict tree to see how wrong the tree is
#
     if(opt.tree) {
        if(seed != 0)
           set.seed(seed + 21)
        out <- cv.tree(big, FUN = prune.tree)
        opt.treesize <- out$size[order(out$dev)[1]]
        if(opt.treesize == 1) {
           opt.treesize <- 2
           cat(" opt tree size changed from 1 to 2")
```

```
         }
         if(verbose) {
            cat("optimal tree size for leave one out cv is ",
                  opt.treesize, "\n")
         }
         avg.opt.treesize <- avg.opt.treesize + opt.treesize
         opt.small <- prune.tree(big, best = opt.treesize)    #
#
# predict  the current optimal small tree, so as to do comparison
# of the tree method versus the knn in leafs
#
         dpred <- predict(opt.small)
         dpred <- apply(dpred, 1, function(x)
         (0:(numClass - 1))[x == max(x)])    #
#
# Handle ugly case where there are ties (and therefore dpred is
# a list)
#
         if(is.list(dpred)) {
            dpred <- sapply(dpred, function(x)
            {
              if(length(x) > 1)
                x[sample(1:length(x), size = 1)]
              else x
            }
            )
         }
         Tree.Sum.Misclass <- sum(dpred != big.data[, 1])
      }
#   end opt.tree if loop
   }
# end if n = 1 loop
#
# all done, tally up and report final results,
#
   avg.opt.k <- avg.opt.k/(treesize * n)
   avg.opt.treesize <- avg.opt.treesize/n
   return(list = c(Misclass = Sum.Misclass, Tree.Misclass =
         Tree.Sum.Misclass, Avg.Opt.Treesize = avg.opt.treesize,
         avg.opt.k = avg.opt.k, Total = nrow(data)))
}
```

2. knn.in.leaf

```
function(trg.set, trg.classes, test.set, test.classes,
treesize = 4, kstep = 1, no.k = 8, seed = 0, opt.tree = T,
opt.tree.size = 0, verbose = F)
{
#
# By Ciril Karo
#
# knn.in.leaf: do knn-in-leaf for test set in tree of size
# stipulated for training set.  Build the tree from training
# set, allocate test items to leafs, conduct leave one out
# validation of training set in leaves to find optimal k, do knn
# in leaf with that k and test set items allocated to that leaf.
# Add up misclassifications.
# If opt.tree is true, build the optimal tree of size specified,
# or of optimal size found by tree.cv().  Predict test set and
# report number of misclassifications
#
# Arguments: trg.set : full trg set of data (without classes)
#       trg.classes  : vector of classes of the trg.set of data
#                       (assume numeric classes in form 0,1,2 etc)
#       test.set     : full test set of data (without classes)
#       test.classes : vector of classes of the test.set of data
#                      (assume numeric classes in form  0,1,2 etc)
#       treesize     : scalar size of tree do knn-in-leaf in.
#       kstep        : steps size for optimal k in leaf search
#       no.k         : number of k for optimal k in leaf search
#       seed         : seed to set if doing Common Random Numbers
#                      work.  If seed=0, does not set seed
#       opt.tree     : whether or not to produce optimal
#                      classification tree (if doing multiple
#                      runs can save work)
#       opt.tree.size: allows classification tree size to be
#                      set(if 0 then defaults to finding optimal
#                      tree size automatically)
#       verbose      : whether optimal tree size and other
#                      information is to be dumped to screen
#
#     return value : sum of misclassifications knn-in-leaf
#                  : sum of misclassifications in optimal tree
#
# The overall Plan for this function:
# 1. Build the tree for trg.set data
# 2. Build optimal tree if required and predict test set
# 3. For each leaf of tree:
#    a. find all test items assigned to that leaf
#    b. find optimal k for leaf using leave one out cv in leaf
#       training items
#    c. do knn for test items using k found, count up
#       misclassifications
# 4.  Report answers
#
# set up some variables and preliminaries
#
   Sum.Misclass <- Tree.Misclass <- 0
   opt.treesize <- avg.opt.k <- 0
   temp.cv <- numeric(no.k)
   big.trg <- cbind(class = trg.classes, trg.set)
   big.trg <- data.frame(big.trg)
   big.test <- cbind(class = test.classes, test.set)
   big.test <- data.frame(big.test)    #
#
# assume that classes are numeric 0,1,2 etc
#
   numClass <- max(trg.classes) + 1
   assign("numClass", numClass, frame = 1)  #
#
# ugly little name switch here, but allows me to reuse some code
#
   temp.data <- big.trg
   tst.data <- big.test
   assign("tst.data", tst.data, frame = 1)
   assign("temp.data", temp.data, frame = 1)
   big <- tree(as.factor(class) ~ ., data = temp.data)
   assign("big", big, frame = 1)
   small <- prune.tree(big, best = treesize)      #
#
# find optimal sized small tree if wanted.  Seed is required so
# same tree selected each time.  Will rarely be required, but
# left in for completeness
#
   if(opt.tree) {
      if(opt.tree.size == 0) {
         if(seed != 0) set.seed(seed)
         out <- cv.tree(big, FUN = prune.tree)
         opt.treesize <- out$size[order(out$dev)[1]]
      }
      else {
         opt.treesize <- opt.tree.size
      }
      if(opt.treesize == 1) {
         opt.treesize <- 2
         cat(" opt tree size changed from 1 to 2 \n")
      }
      if(verbose) {
         cat("optimal tree size is ", opt.treesize, "\n")
      }
      opt.small <- prune.tree(big, best = opt.treesize)#
#
# predict  the current optimal small tree, so as to do comparison
# of the tree method versus the knn in leafs
```

```
#
      dpred <- predict(opt.small, newdata = tst.data[, -1])
      dpred <- apply(dpred, 1, function(x)
             (0:(numClass - 1))[x == max(x)]) #
#
# Handle ugly case where there are ties (and therefore dpred
# is a list)
#
      if(is.list(dpred)) {
         dpred <- sapply(dpred, function(x)
         {
            if(length(x) > 1)
              x[sample(1:length(x), size = 1)]
            else x
         }
         )
      }
      Tree.Misclass <- sum(dpred != tst.data[, 1])
   }
# end if (opt.tree)
#
# from small I want the leaf node numbers.  This gets them for me
#
   leafs <- as.numeric(dimnames(small$frame)[[1]]
         [small$frame[, "var"] == "<leaf>"]) #
#
# now predict which leaf each chunk element will be in using
# predict.leaf written by Sam Buttrey
#
   chunk.leaf <- predict.leaf(small, tst.data[, -1])    #
#
# time to loop over each leaf in turn to do knn-in-leaf
#
   for(j in 1:treesize) {
#
# identify the elements in the leaf
#
      sub.leaf <- temp.data[identify(small, leafs[j]),  ]
      tst.leaf <- tst.data[chunk.leaf == leafs[j],  , drop = F]
      if(nrow(tst.leaf) == 0) {
         cat(" 0 rows in test\n")
         cat(nrow(sub.leaf), " rows in trg\n")
         next
      }
      if(verbose) {
         cat("doing knn in leaf, leaf number ", leafs[j], " \n")
         cat(" size leaf.trg ", nrow(sub.leaf), "\n")
         cat(" size leaf.test ", nrow(tst.leaf), "\n")
      }
#
# now do cross validation in each leaf to determine optimal k
# note this is Ripley's leave one out cv function
#
      for(k in 1:no.k) {
         if(kstep > 1) {
            temp.cv[k] <- sum(sub.leaf[, 1] !=
                 knn.cv(sub.leaf[, -1], sub.leaf[, 1],
                 kstep * k - 1))
         }
         else {
            temp.cv[k] <- sum(sub.leaf[, 1] !=
                 knn.cv(sub.leaf[, -1], sub.leaf[, 1], k ))
         }
      }
#  end for loop over number of k steps
      opt.k <- order(temp.cv)[1]
      avg.opt.k <- avg.opt.k + opt.k
      if(verbose) {
         if(kstep > 1) {
            cat("opt k in leaf ", leafs[j], " is ",
                 kstep * opt.k - 1, " \n")
         }
         else {
            cat("opt k in leaf ", leafs[j], " is ", opt.k, " \n")
         }
         if(opt.k == no.k)
            cat(" opt.k = no.k, so you may want to increase k \n")
      }
#
# use Optimal k to get misclassifications for test sample and add
# to misclass sum.
#
      if(kstep > 1) {
         Sum.Misclass <- Sum.Misclass + sum(knn(sub.leaf[, -1],
              tst.leaf[, -1, drop = F], sub.leaf[, 1],
              kstep * opt.k - 1) != tst.leaf[, 1])
      }
      else {
         Sum.Misclass <- Sum.Misclass + sum(knn(sub.leaf[, -1],
              tst.leaf[, -1, drop = F], sub.leaf[, 1], opt.k) !=
              tst.leaf[, 1])
      }
   }
# this brace is closing loop over leafs
# all done, tally up and report final results
#
   avg.opt.k <- avg.opt.k/treesize
   return(list = c(knn.in.leaf.Misclass = Sum.Misclass,
         Tree.Misclass = Tree.Misclass, Opt.Treesize =
         opt.treesize, avg.opt.k = avg.opt.k,
         Total = nrow(test.set)))
}
```

3. predict.leaf

```
function(tree, new) {

# by Sam Buttrey
#
# Predict.leaf: second cut at function to return a vector
# of leaf numbers for new input data into a tree.
#
# Allow for the case where "new" is a vector
# Also change a matrix to a data.frame if need be
   if(is.null(nrow(new))) new <- data.frame(matrix(new,
         nrow = 1))
   if(!is.data.frame(new)) new <- data.frame(new) #
#
# Call the predict function with type = "tree".
#
   out <- predict(tree, new, type = "tree") #
#
# Out$where names the *row* of the frame matrix in which each
# leaf can be found. (Note that this row's number is different
# from the node's number.) So grab those...
#
   return(as.numeric(dimnames(out$frame)[[1]][out$where]))   #
}
```

C. r.d FUNCTIONS

1. sams.knn.rd.v3()

```
function(trng, trng.classes, lower, upper, nr, bestonly = T)
{
#
# By Sam Buttrey and Ciril Karo
#
# sams.knn.rd.v3: Function to find a good r.d, that is, threshold
# distance for considering nn's, by cross-validating on the
# training set. This is a leave-one-out cross-validation.
# .v3 explicitly allows 1 nn in all r.d's rather than counting as
# a misclass.  That is if no neighbors fall inside a distance,
# then algorithm defaults to 1NN.
#
# Tips for use - start with lower set to zero and upper high,
# search any distances that appear to offer improvements.
# Zooming in can help.  Note this is highly non-linear and
# multiple minima can exist.
#
# Args: trng: training set
#     classes: Classes of training set
#     lower  : lower limit of r.d search region
#     upper  : upper limit of r.d search region
#          nr: Number of r.d's to consider
#   bestonly: Other functions only require the best r.d distance
#             this switch supplies only that answer when true
#
# Return value: vector of r.d's (length nr) and vector (same
# length) of cross validation misclassification sums.  If
# bestonly , only the best r.d distance is returned where best is
# defined as the one with lowest number of misclassifications
#
# Check to see "daisy" exists. If not, get it.
#
   if(!exists("daisy")) library(cluster)    #
#
# Create results.
#
   misclass.sum <- numeric(nr)    #
   all.dists <- daisy(trng)
   k.mat <- matrix(0, nrow(trng), nr)
   dist.mat <- matrix(0, nrow(trng), nrow(trng)) #
#
# Okay. Daisy gives us a weird vector in which there is one
# distance from each row to the ones farther down in the data
# frame. So the distances for row 1 extend from 1 to (n-1), where
# n = nrow(trng); the distances for row i extend from (end of
# those for (i-1)) to (end of those for(i-1), plus n, minus i).
# The ones we want are the last "chunk" of those; those are the
# distances to members of this training set chunk.
#
   nrow.big.data <- nrow(trng)
   for(i in 1:(nrow.big.data - 1)) {
      if(i == 1) {
         start <- 1
         finish <- nrow.big.data - 1
      }
      else {
         start <- finish + 1
         finish <- finish + nrow.big.data - i
      }
      dist.mat[(i + 1):nrow(trng), i] <- dist.mat[i,
            (i + 1):nrow(trng)] <- all.dists[start:finish]
   }
#
# Apply "order" to each row. This will come out in columns.
# Transpose to get back to rows; then remove the first column,
# which is "i" for row i.
#
   orders <- t(apply(dist.mat, 1, order))[, -1]
   dists <- t(apply(dist.mat, 1, sort))[, -1]     #
```

```
#
   r.d <- seq(lower, upper, length = nr)
   for(i in 1:nrow(trng)) {
      for(j in 1:nr) {
         contenders <- orders[i,  ][dists[i,  ] <= r.d[j]]
         classes <- trng.classes[contenders]
         if(length(classes) == 0) {
#
# In this version we explicitly reset 0 contender back to 1NN.
#
            contenders <- orders[i,  ][1]
            classes <- trng.classes[contenders]
         }
         y <- table(classes)
         k.mat[i, j] <- length(classes)
         winner <- as.numeric(names(y)[y == max(y)])#
#
# Handle ties randomly
#
if(length(winner) > 1)
            decision <- winner[sample(1:length(winner), size = 1)]
         else decision <- winner
         if(decision != trng.classes[i])
            misclass.sum[j] <- misclass.sum[j] + 1
      }
   }
   avg.k.r.dist <- mean(k.mat[, order(misclass.sum)[1]])
   if(bestonly) {
      r.Dist <- r.d[order(misclass.sum)[1]]
      return(r.Dist)
   }
   else {
      return(list(Misclass.Sum = misclass.sum, r.Distances = r.d,
            avg.k.best.r.dist = avg.k.r.dist))
   }
}
```

2. knn.rd.cv

```
function(data, classes, n = 12, lower.rd, upper.rd, nr = 20,
verbose = F) {
#
# By Ciril Karo
#
# knn.rd.cv: Do n-fold cross-validation on the full data using
# r.d.  See plan below:
#
# Arguments: data  : full of data to be used in cross validation
#                     (without classes)
#           classes : vector of classes of the full set of data
#                     (assume numeric 0,1,2 etc)
#           n       : number of chunks to split the data into
#                     (They won't necessarily be exactly the
#                     same size.)
#           lower.rd: lower limit on range of rd to search over
#           upper.rd: upper limit on range of rd to search over
#                     (these should be found in preliminary work
#                      with sams.knn.rd.v3)
#           nr      : number of r.d values to consider between
#                     lower and upper
#
# return value: sum of misclassifications
#               average number of k considered
#
# Here is a  plan of attack (this will be slow)
#
# 1. Chunk the data - n = 12 is becoming standard
# 2. new function "sams.knn.rd.v3" will do "leave one out" cross
#    validation on a range of r.d values whose limits are
#    supplied as lower and upper by user and return the optimal
#    r.d value.  Some investigation on the range could be done
#    using sams.knn.rd.v3 (manually for now).
# 3. Now we have test chunk (1/12) and a trg chunk (11/12).   Get
#    100nn for the test portion using "new.sams.knn()"
# 4. Now sum the $dists <= r.d to give the k to use.
# 5. Finally plunk the $Classes and k vector into "majority.big".
#    This will return a vector of classes to compare to actual
#    classes for the test piece - add up misclassifications.
# 6. Goto 2, do the same for the next test chunk.
#
# Get the sample. Set up the vector of places where the chunks
# start as a vector of evenly-spaced non-integers; then round
# 'em. The last one is exactly nrow(data); that's okay.
#
   samp <- sample(1:nrow(data))
   chunk.start <- round(seq(1, nrow(data), len = n + 1))
         [ - (n + 1)]  #
#
# Loop through the chunks. Each chunk goes from its starting
# point to one less than the next starting point (except that the
# last chunk includes the final data point.) For each chunk, do
# stuff listed above.  First of all some variable set up stuff
#
   Sum.Misclass <- avg.opt.k <- 0
   big.data <- cbind(class = classes, data)
   big.data <- data.frame(big.data)    #
#
# this is cheating a bit, assumes that classes are numeric 0,1,2
# etc maybe fix later, nope!
#
```

```
   numClass <- max(classes) + 1
   assign("numClass", numClass, frame = 1)
   for(i in 1:n) {
      if(i == n) out <- samp[(chunk.start[i]):nrow(data)]
      else out <- samp[(chunk.start[i]):(chunk.start[i + 1] - 1)]
      trng.bit <- big.data[ - out,  ]
      test.bit <- big.data[out,  ]  #
#
# get the optimal r.d
#
      this.r.d <- sams.knn.rd.v3(trng.bit[, -1], trng.bit[, 1],
           lower.rd, upper.rd, nr, bestonly = T)
      assign("this.r.d", this.r.d, frame = 1)     #
#
# now stage 3, get 100nn.  100 NN is arbitrary, could be set
# higer or lower if required.
#
      this.100nn <- new.sams.knn(trng.bit[, -1], test.bit[, -1],
           trng.bit[, 1], 100)    #
#
# Now I need to check if there are any NA in the dists and if so
# replace with large number.  IE NA due to less than 100 items in
# set, thus we will have errors later if no number.  But setting
# to above r.d value fixes all up.
#
      this.100nn$Dists[is.na(this.100nn$Dists)] <- (this.r.d + 1)
      num.k <- apply(this.100nn$Dists, 1, function(x)
           sum(x <= this.r.d))   #
#
# stage 5
#
      num.k[num.k == 0] <- 1
      aver.k <- mean(num.k)
      c.size <- length(out)
      Sum.Misclass <- Sum.Misclass + sum(test.bit[, 1] !=
           majority.big(this.100nn$Classes, num.k))    #
      if(verbose)
         cat("endchunk  ", i, "  r.d  ", this.r.d, " SumMisclass
              ", Sum.Misclass, "average k ", aver.k, "\n")
      avg.opt.k <- avg.opt.k + aver.k
#
# now loop to next chunk and do it all again
#
   }
   avg.opt.k <- avg.opt.k/n #
#
# all done, report final results
#
   return(list = c(Misclass = Sum.Misclass,
        Avg.Opt.k = avg.opt.k, Total = nrow(data)))
}
```

3. Example of the S-Plus Code to Apply r.d to a Test Set.

The code below is an example of how to apply r.d to a test set using the functions available. It has not been written into a function as it was only used a few times and evidence from the results was that it would not be used again. The data set is Image.

This is the code to find the best r.d distance for the training set. Examining a range of distances between lower and upper with variations of granularity (setting nr) is important. The output here is the distances and the number of misclassification so it is simple to scan for the best distance.

```
sams.knn.rd.v3(image.trg[,-1], image.trg[,1],lower=13, upper=15,
      nr=20, bestonly=F)
```

Once satisfied that you have zoomed in on the optimal distance range, this code will assign the best distance ready to be used.

```
best.r.d<- sams.knn.rd.v3(image.trg[ ,-1], image.trg[ ,1],
      lower = 14.3, upper = 15, nr = 20, bestonly = T)
```

To actually find the number of nearest neighbors you need the output from new.sams.knn(). In here there will be a number n of nearest neighbors (100 usually), along with the distances to these 100 neighbors. This code makes the 100NN object.

```
image.test.100nn <- new.sams.knn(image.trg[ ,-1],
      image.test[ ,-1], image.trg[ ,1], 100)
```

The 100NN object has distances to each test item The best r.d is known. The two together in this code finds the vector of k's to feed to the majority.big NN voter.

```
num.k<-apply(image.test.100nn$Dists,1,function(x)
      sum(x <= best.r.d))
```

Finally classify the test set using majority.big which uses the classes matrix from the 100NN object and the vector num.k of k's to be used. Add up the misclassifications.

```
sum(majority.big (image.test.100nn$Classes, num) !=
      image.test[ ,1])
```

APPENDIX B. RAW RESULTS

The results below are reported by data sets. Each data set was examined in a raw un-scaled form and in a scaled form where each variable was normalized to mean 0 and standard deviation 1. For each data set the following is a description of what will be reported:

1. Data set statistics including size of data set, number of classes, along with a description of what the class represents, and number of independent variables.

2. n-fold cross-validation results, leave-one-out cross-validation results or test set classification results for each method as follows:

a. **Raw Misclass.** The mean number of misclassifications for the method. This is a random variable from the Monte Carlo replications. For the leave-one-out and test set classifications the number is essentially constant. There may be small variations due to the random tie-break mechanism.

b. **Std Dev.** Standard Deviation for the Raw Misclassifications in the n-fold cross-validation.

c. **Replications.** Number of Monte Carlo replications in the n-fold cross validation, usually done with Common Random Numbers (exceptions noted).

d. **p-value.** Testing the null hypothesis that the means of the misclassifications from the Monte Carlo replications are the same. From a paired t-test of the means of the new method (knn-in-leaf or r.d) versus the best of the baseline methods for n-fold cross validation.

e. **Misclass Rate and SD as Rate.** Raw Misclassification and Standard Deviation as percentage of full data set.

f. **Tree Size.** This has two meanings depending on method. It is the average optimal tree size for the classification tree as chosen by cross-validation and the minimization of residual deviance. In knn-in-leaf it is the tree size which produced lowest mean misclassifications.

g. **Average k.** Average k used in *k*-NN and r.d. Occasionally the average *k* used in the knn-in-leaf tree was also captured. This is an average across all leaves. In the test set classification and leave-one-out results this is just one observation.

h. **Rank.** Ranks the methods from 1 to 4 based on the Misclassification Rate.

A. DIABETES DATA

This data set contains 768 instances, There are 500 of class 0 and 268 of class 1. Each instance has 8 continuous variables. Class 1 indicates the subject has diabetes. Table 8 and 9 contain the raw results for un-scaled and scaled Diabetes data.

Un-scaled Data	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
12 Fold Cross-validation					768 items
Raw Misclass	196.3	201.5	180.2	190.3	
Std Dev	5.3	8.5	9.8	4.2	
Replications	10	10	10	10	
p-value			0.0002	0.002	vs knn
Misclass rate	25.6%	26.2%	23.5%	24.8%	
SD as rate	0.7%	1.1%	1.3%	0.5%	
Tree size	-	6.2 leaf	9 leaf	-	
Average *k*	18.6	-	-	76	Note 1
Rank	3	4	1	2	
Leave-one-out Cross-validation					
Raw Misclass	184	175	129	185	
Misclass Rate	24.0%	22.8%	16.8%	24.1%	
Tree size	-	7	9	-	
Average *k*	19	-	9.2	114	Note 1
Rank	3	2	1	4	

Table 8. Raw Results for Diabetes Data

1. Number of *k* used in r.d is free in leave-one-out cross-validation, but set at a maximum of 100 for n-fold cross-validation.

Scaled Data	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
12 Fold Cross-validation					768 items
Raw Misclass	191.6	201.5	186.9	203.2	
Std Dev	4.6	8.5	8.4	5.9	
Replications	10	10	10	10	
p-value			0.114	N/A	vs knn
Misclass rate	24.9%	26.2%	24.3%	26.5%	
SD as rate	0.6%	1.1%	1.1%	0.8%	
Tree size	-	6.2	9	-	
Average *k*	21.6	-	4	59.4	
Rank	2	3	1	4	
Leave-one-out Cross-validation					
Raw Misclass	184	175	131	187	
Misclass Rate	24.0%	22.8%	17.2%	24.3%	
Tree size	-	7 leaf	9	-	
Average *k*	23	-	6	74.9	
Rank	3	2	1	4	

Table 9. Raw Results for Scaled Diabetes Data

B. VEHICLE RECOGNITION DATA

This data set contained 846 items, each with 18 continuous variables. There were four classes representing the silhouettes of vehicles. Each class was approximately equally represented in the data set. Table 10 contains the raw results for the un-scaled data and Table 11 the results for the scaled data.

Un-scaled Data	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
12 Fold Cross-validation					846 items
Raw Misclass	302.1	249.8	239.7	297.8	
Std Dev	8.8	9.4	11.2	6.1	
Replications	10	10	10	5	Note 1
p-value			0.025	N/A	vs class tree
Misclass rate	35.7%	29.5%	28.3%	35.2%	
SD as rate	1.0%	1.1%	1.3%	0.7%	
Tree size	-	16.2	17	-	
Average *k*	3.5	-	-	6.6	
Rank	4	2	1	3	
Leave-one-out Cross-validation					
Raw Misclass	281	199	178	278	
Misclass Rate	33.2%	23.5%	21.0%	32.9%	
Tree size	-	16	17	-	
Average *k*	3	-	3.6	6.1	
Rank	4	2	1	3	

Table 10. Raw Results for Vehicle Recognition Data

1. *k*-NN and r.d replications did not use Common Random Numbers.

Scaled Data	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
12 Fold Cross-validation					846 items
Raw Misclass	237.6	249.8	229.8	251.7	
Std Dev	8.8	9.4	8.3	8.06	
Replications	10	10	10	10	
p-value			0.033	N/A	vs knn
Misclass rate	28.1%	29.5%	27.2%	29.8%	
SD as rate	1.0%	1.1%	1.0%	1.0%	
Tree size	-	16.2	17	-	
Average *k*	5.5	--	3	15	
Rank	2	3	1	4	
Leave-one-out Cross-validation					
Raw Misclass	226	199	168	233	
Misclass Rate	26.7%	23.5%	19.9%	27.5%	
Tree size	-	16	17	-	
Average *k*	6	-	3.8	17	
Rank	3	2	1	4	

Table 11. Raw Results for Scaled Vehicle Recognition Data

C. SONAR DATA

This data is distinguishing between rocks and mines on basis of sonar responses. There are 208 instances, each with 60 variables. There are 98 instances of class 1 (mines) and 110 of class 0 (rocks). The results for the data set are in Table 12, with the results for the scaled data in Table 13.

Un-scaled data	1-NN	*k*-NN	Class Tree	knn-in-leaf	r.d	Remarks
10 Fold Cross-validation						208 items
Raw Misclass	36.2	39.5	57.3	40.3	36.3	
Std Dev	1.2	2.5	2.0	2.8	1.2	
Replications	10	10	10	10	10	
p-value				N/A	N/A	
Misclass rate	17.4%	19.0%	27.5%	19.4%	17.5%	
SD as rate	0.6%	1.2%	1.0%	1.3%	0.6%	
Tree size	-	-	3	2.5	-	
Average *k*	1	1.8	-	-	1	
Rank	1	2	3	2	1	
Leave-one-out Cross-validation						
Raw Misclass	36	36	47	27	36	
Misclass Rate	17.3%	17.3%	22.6%	13.0%	17.3%	
Tree size		-	4	4	-	
Average *k*	1	1 or 5	-	4	1.5	
Rank	2	2	3	1	2	

Table 12. Raw Results for Sonar Data

Scaled Data	1-NN	*k*-NN	Class Tree	knn-in-leaf	r.d	Remarks
10 fold Cross Validation						208 items
Raw Misclass	26.5	29.5	57.3	32.6	26.5	
Std Dev	2.4	2.4	2.0	4.9	2.4	
Replications	10	10	10	10	10	
p-value				N/A	N/A	
Misclass rate	12.7%	14.2%	27.5%	15.7%	12.7%	
SD as rate	1.2%	1.2%	1.0%	2.4%	1.2%	
Tree size	-	-	2.5	3	-	
Average *k*	1	1.3	-	2.7	1.3	
Rank	1	2	4	3	1	
Leave-one-out Cross-validation						
Raw Misclass	26	26	47	23	25	
Misclass Rate	12.5%	12.5%	22.6%	11.1%	12.0%	
Tree size	-	-	4	4	-	
Average *k*	1	1	-	2.5	1.7	
Rank	3	3	4	1	2	

Table 13. Raw Results for Scaled Sonar Data

D. IMAGE SEGMENTATION DATA

This data set consists of a training set of 210 instances and a test set of 2100 instances. There are seven classes equally represented in each set, representing image details for a range of outdoor surfaces, e.g. grass, foliage, concrete, brickwork, etc. There are 19 continuous attributes per instance. Table 14 contains the raw results for the un-scaled data and Table 15 for the scaled case.

Un-Scaled	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
Classification of test set					2100 items
Raw Misclass	259	237	183	269 (259)	Note 1
Misclass Rate	12.3%	11.3%	8.7%	12.3%	
Tree size	-	10	13	-	
Average *k*	1	-	1.7	1.3 ->1	Note 1
Rank	3	2	1	3	
Leave-one-out Cross-validation					210 items
Raw Misclass	33	11	10	33	
Misclass Rate	15.7%	5.2%	4.8%	15.7%	
Tree size	-	10	10	-	
Average *k*	1	-	1.3	1 or so	
Rank	3	2	1	3	

Table 14. Raw Results for Image Segmentation Data

1. r.d approached the 1-NN result, due to the algorithm defaulting to 1-NN if the distance was so small that no nearest neighbors were included.

Scaled data	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
Classification of Test Set					2100 items
Raw Misclass	200	237	183	200	
Misclass Rate	9.5%	11.3%	8.7%	9.5%	
Tree size		10	7		
Average *k*	1	-	1.4	1	
Rank	2	3	1	2	
Leave-one-out Cross-validation					210 items
Raw Misclass	26	11	9	30	
Misclass Rate	12.4%	5.2%	4.3%	14.3%	
Tree size	-	10	10	-	
Average *k*	6	-	1.9	1	
Rank	3	2	1	4	

Table 15. Raw Results for Scaled Image Segmentation Data

E. WAVES DATA

This data set consists of a generated training set of 300 instances and a generated test set of 3000 instances. There are three classes approximately equally represented in each set. There are 21 continuous attributes per instance. The raw results for the un-scaled data are in Table 16 and for the scaled data in Table 17.

Un-Scaled	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
Classification of Test Set					3000 items
Raw Misclass	525	1121	663	509	
Misclass Rate	17.5%	37.4%	22.1%	17.0%	
Tree size	-	5	4	-	
Average *k*	15	-	7.4	31.6	
Rank	2	4	3	1	
Leave-one-out Cross-validation (Training Set)					300 items
Raw Misclass	45	84	44	40	
Misclass Rate	15.0%	28.0%	14.7%	13.3%	
Tree size	-	5	3	-	
Average *k*	15	-	7	31.6	
Rank	2	4	3	1	

Table 16. Raw Results for Waves Data

Scaled data	*k*-NN	Classification Tree	knn-in-leaf	r.d	Remarks
Classification of Test Set					3000 items
Raw Misclass	501	1121	672	524	
Misclass Rate	16.7 %	37.4%	22.4%	17.5%	
Tree size	-	5	3	-	
Average *k*	21	-	11	55.2	
Rank	1	4	3	2	
Leave-one-out Cross-validation (Training set)					300 items
Raw Misclass	51	84	44	45	
Misclass Rate	17.0%	28.0%	14.7%	15.0%	
Tree size		5	7		
Average *k*	21/11	7	4.8	56	
Rank	3	4	1	2	

Table 17. Raw Results for Scaled Waves Data

LIST OF REFERENCES

Bailey T., and Jain A.K., "A Note on Distance Weighted *k*-Nearest Neighbor Rules," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC 8, No 4, April 1978, pp. 311-313.

Breiman L., Friedman J., Olshen R., and Stone C., *Classification and Regression Trees*, Monterey CA, 1984.

Broder A.J., "Strategies for Efficient Incremental Nearest Neighbor Search," *Pattern Recognition Letters*, Vol. 23, No. 1/2, January/February 1990, pp. 171-178.

Brown T.A., and Koplowitz J., "The Weighted Nearest Neighbor Rule for Class Dependant Sample Sizes," *IEEE Transactions on Information Theory*, Vol. IT 25, No. 5, September 1979, pp. 617-619.

Cover T.M., and Hart P.E., "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, Vol. IT-13, No. 1, pp. 21-27, January 1967.

Cover T.M., "Rates of Convergence for Nearest Neighbor Procedures," *Proceedings 1st Annual Hawaii Conference of Systems Theory*, pp. 413-415, January 1968.

Dasarathy B.V., and Sheela B.V., "A Composite Classifier System Design: Concepts and Methodology," *Proceedings of the IEEE (Special Issue on Pattern Recognition and Image Processing)*, Vol. 67, No. 5, pp. 708-713, May 1979.

Dasarathy B.V., (Ed), *Nearest Neighbor Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp.1-22.

Dudani S.A., "The Distance Weighted *k*-Nearest Neighbor Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC 6, No. 4, April 1976, pp. 325-327.

Fix E., and Hodges J.L., *Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties*, Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Texas 1951, reprinted in Dasarathy 1990.

Fukunaga K., and Narendra P.M., "A Branch and Bound Algorithm for Computing *k*-Nearest Neighbors," *IEEE Transactions on Computers*, Vol. C-24, No 7, pp. 750-753, July 1975.

Gorman, R.P., and Sejnowski, T.J., "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," *Neural Networks*, Vol. 1, pp. 75-89, 1988.

Macleod J.E.S., Luk A., and Titterington D.M., "A Re-Examination of the Distance Weighted *k*-Nearest Neighbor Classification Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC 11, No. 4, July/August 1987, pp. 689-696.

Merz C., and Murphy P., *U. C. I. Repository of Machine Learning Databases [http://www.ics.uci.edu/mlearn/MLRepository.html]*, Irvine CA: University of California, Department of Information and Computer Science, 1996.

Michie D., Spiegelhalter D., and Taylor C., *Machine Learning , Neural and Statistical Classification*, New York: Ellis Horwood, 1994.

Morin R.L., and Raeside D.E., "A Reappraisal of Distance Weighted *k*-Nearest Neighbor Classification for Pattern Recognition with Missing Data," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC 11, No. 3, March 1981, pp. 241-243.

O'Callaghan J.F., "An Alternative Definition for Neighborhood of a Point," *IEEE Transactions on Computers*, Vol. C-24, No. 11, pp. 1121-1125, November 1975.

Patrick E.A., and Fisher F.P. III., "A Generalized *k*-Nearest Neighbor Rule," *Information and Control*, Vol. 16, pp. 128-152, April 1970.

Peterson D.W., "Some Convergence Properties of a Nearest Neighbor Decision Rule," *IEEE Transactions on Information Theory*, Vol. IT-16, No. 1, pp. 26-31, January 1970.

Siebert J.P., *Vehicle Recognition Using Rule Based Methods*, Turing Institute Research Memorandum TIRM-87-018, March 1987.

Short R.D., and Fukunaga K., "The Optimal Distance Measure for Nearest Neighbor Classification," *IEEE Transactions on Information Theory*, Vol. IT-27, No. 5, pp.622-627, September 1981.

Smith J.W., Everhart J.E., Dickson W.C., Knowler W.C., and Johannes R.S., "Using the ADAP learning algorithm to forecast the onset of diabetes mellitus," *Proceedings of the Symposium on Computer Applications and Medical Care.* IEEE Computer Society Press, pp. 261--265, 1988.

S-PLUS 4 Guide to Statistics, Data Analysis Products Division, MathSoft, Seattle, 1997.

StatLib - Software and extensions for the S (S-Plus) language [http://lib.stat.cmu.edu/S/] (for down loaded S-Plus classification library).

Tomek I., "A Generalization of the *k*-NN Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-6, No. 2, pp. 121-126, February 1976.

Venables W.N., & Ripley B.D., *Modern Applied Statistics with S-Plus*, Springer, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor Samuel E. Buttrey 1
   Code OR/SB
   Naval Postgraduate School
   Monterey California 93943-5002

4. Professor Harold J. Larson 1
   Code OR/LA
   Naval Postgraduate School
   Monterey California 93943-5002

5. Director.. 1
   Capability Development Wing
   Tobruk Barracks
   Puckapunyal, Victoria 3662
   Australia

6. Major Ciril Karo.. 2
   Capability Development Wing
   Tobruk Barracks
   Puckapunyal, Victoria 3662
   Australia